ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INFORMÁTICOS

Titulación:

GRADO EN INGENIERÍA INFORMÁTICA

Título del Proyecto:

SISTEMA DE MONITORIZACIÓN DE SISTEMAS DISTRIBUÍDOS

Alumno: Alberto Vilas Virulegio

Tutor: Ricardo Jiménez Péris

Boadilla del Monte, 6 de junio de 2014

Resumen

La computación distribuida ha estado presente desde hace unos cuantos años, pero es quizás en la actualidad cuando está contando con una mayor repercusión. En los últimos años el modelo de computación en la nube (Cloud computing) ha ganado mucha popularidad, prueba de ello es la cantidad de productos existentes. Todo sistema informático requiere ser controlado a través de sistemas de monitorización que permiten conocer el estado del mismo, de tal manera que pueda ser gestionado fácilmente. Hoy en día la mayoría de los productos de monitorización existentes limitan a la hora de visualizar una representación real de la arquitectura de los sistemas a monitorizar, lo que puede dificultar la tarea de los administradores. Es decir, la visualización que proporcionan de la arquitectura del sistema, en muchos casos se ve influenciada por el diseño del sistema de visualización, lo que impide ver los niveles de la arquitectura y las relaciones entre estos. En este trabajo se presenta un sistema de monitorización para sistemas distribuidos o Cloud, que pretende dar solución a esta problemática, no limitando la representación de la arquitectura del sistema a monitorizar. El sistema está formado por: agentes, que se encargan de la tarea de recolección de las métricas del sistema monitorizado; un servidor, al que los agentes le envían las métricas para que las almacenen en una base de datos; y una aplicación web, a través de la que se visualiza toda la información. El sistema ha sido probado satisfactoriamente con la monitorización de CumuloNimbo, una plataforma como servicio (PaaS), que ofrece interfaz SQL y procesamiento transaccional altamente escalable sobre almacenes clave valor. Este trabajo describe la arquitectura del sistema de monitorización, y en concreto, el desarrollo de la principal contribución al sistema, la aplicación web.

Abstract

Distributed computing has been around for quite a long time, but now it is becoming more and more important. In the last few years, cloud computing, a branch of distributed computing has become very popular, as its different products in the market can prove. Every computing system requires to be controlled through monitoring systems to keep them functioning correctly. Currently, most of the monitoring systems in the market only provide a view of the architectures of the systems monitored, which in most cases do not permit having a real view of the system. This lack of vision can make administrators' tasks really difficult. If they do not know the architecture perfectly, controlling the system based on the view that the monitoring system provides is extremely complicated. The project introduces a new monitoring system for distributed or Cloud systems, which shows the real architecture of the system. This new system is composed of several elements: agents, which collect the metrics of the monitored system; a server, which receives the metrics from the agents and saves them in a database; and a web application, which shows all the data collected in an easy way. The monitoring system has been tested successfully with Cumulonimbo. CumuloNimbo is a platform as a service (PaaS) which offers an SQL interface and a high-scalable transactional process. This platform works over key-value storage. This project describes the architecture of the monitoring system, especially, the development of the web application, which is the main contribution to the system.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

INTRODUCCIÓN

Este trabajo se enmarca dentro de los sistemas de monitorización, más concretamente sobre la monitorización de un sistema escalable de gestión de datos en cloud. Este sistema se caracteriza por estar formado por muchos nodos o subsistemas con distintos roles, y a su vez múltiples instancias por rol, lo que nos lleva hacia una jerarquía. Como consecuencia de estas características, la mayoría de los sistemas de monitorización existentes no pueden dar soporte a su correcta representación y visualización, para poder así monitorizar el sistema de una manera adecuada. La mayoría de las representaciones que se proporcionan son planas, por lo que la gestión de un sistema jerárquico, mediante una visualización plana, se convierte en una ardua tarea para los administradores.
Por ejemplo, en el caso de uno de los sistemas de monitorización más empleados como es Hyperic[1], la disposición del sistema a monitorizar que muestra es la que se representa en la figura 1.1.

Plataformas (nivel físico)	Servidores (nivel lógico)	Servicios (métricas)
Nodo 1 Nodo 2	Subsistema 1 Subsistema 2 Subsistema 3	Métrica 1 Metrica 2 Métrica 3 Métrica 4 ... Métrica N

Figura 1.1: Disposición habitual de los sistemas actuales de monitorización

El propio sistema de monitorización nos lleva a restringir nuestro sistema a una jerarquía de tres niveles, lo que limita sobre todo el nivel lógico, el cual puede estar compuesto por muchos más niveles. Además, la visión de cada uno de los apartados es plana, por lo que la relación entre los niveles físico, lógico y métricas de estos, requiere de un profundo conocimiento del sistema para poder ser gestionada [14].
Dado que Hyperic u otros sistemas similares no nos permiten ver de forma simple la arquitectura del sistema que queremos monitorizar, surge la necesidad de realizar un sistema de monitorización. Este sistema debe permitir que cualquier otro, independientemente de su

[1] http://www.hyperic.com/

arquitectura, pueda ser monitorizado mediante una visualización que mantenga su arquitectura real junto a sus relaciones. Además de que por cada nivel de dicha arquitectura, se presenten métricas del sistema que permitan su monitorización. Una aproximación sería el diagrama mostrado en la figura 1.2.

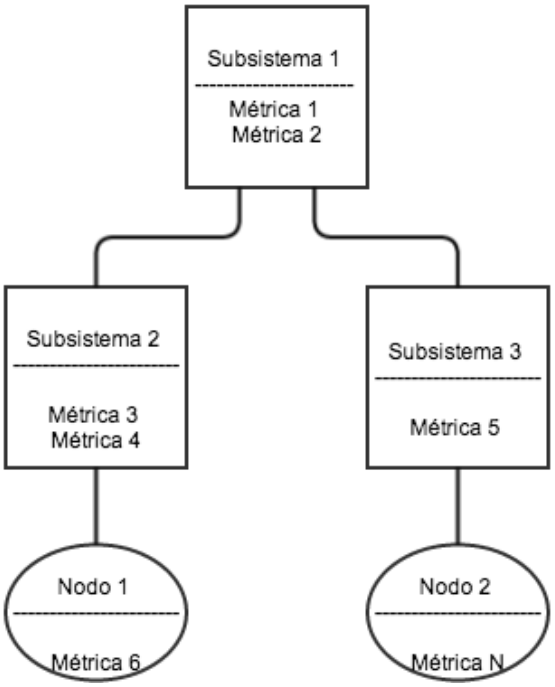

Figura 1.2: Visión que debería de proporcionar el sistema de monitorización

Un sistema de monitorización es el encargado de recolectar métricas de los distintos componentes de un sistema, para su visualización y control. Los valores de dichas métricas permitirán controlar el sistema y llevar a cabo las acciones adecuadas para la gestión y administración del mismo. En este caso, el sistema consta de tres tareas básicas para la monitorización:

- Recolección de las métricas de cada uno de los componentes del sistema que se seleccionen.
- Recepción de dichas métricas enviadas por los recolectores, y almacenamiento de estas para su consulta posterior.
- Visualización del sistema a monitorizar, junto a las métricas que se recolectan de dicho sistema.

El objetivo principal de este trabajo es realizar la parte de visualización del sistema de monitorización que se ha propuesto. Este objetivo se puede dividir en dos apartados. El primero es la conexión de la aplicación con el resto del sistema. El segundo la realización de la parte visual de la aplicación.

Con respecto al primer objetivo, la aplicación permitirá la visualización de la arquitectura, independientemente del sistema, a todos los niveles (lógicos y físico), así como las relaciones entre ellos. Además se deben de poder obtener métricas con información de cada uno de los niveles de los que conste la arquitectura.
Con respecto al segundo objetivo, la aplicación web debe poderse ejecutarse en cualquier navegador, eliminando así cualquier impedimento de desarrollo para una plataforma concreta. Las dos principales funcionalidades que debe de implementar la aplicación a nivel de visualización son las siguientes:

- Un navegador de recursos que permita navegar por los distintos niveles del sistema a monitorizar, y muestre las métricas para dichos niveles. Habitualmente estas métricas se mostrarán en forma de gráficas.
- Un dashboard configurable, al cual se puedan añadir métricas en forma de gráfica, para ser monitorizadas a modo de panel de control.

Para finalizar, se integrará el sistema de monitorización con la plataforma de Big Data del Laboratorio de Sistemas Distribuidos[2], de la Facultad de Informática[3], de la UPM[4]. Este sistema es CumuloNimbo[5]. El objetivo será el de comprobar el funcionamiento y rendimiento del sistema para un caso de uso real.

[2]http://lsd.ls.fi.upm.es/lsd
[3]http://www.fi.upm.es/
[4]http://www.upm.es/
[5]http://www.cumulonimbo.eu/

Capítulo 2

ESTADO DE LA TÉCNICA

En este capítulo se va a dar una visión acerca de los frameworks de programación web actuales para el desarrollo de aplicaciones web. Para comenzar, se definirá qué son las aplicaciones web, y se hará una breve introducción acerca de porqué se ha evolucionado hacia dichas aplicaciones. A continuación, se justificará porque se ha elegido este tipo de sistemas para el desarrollo de la parte gráfica del sistema de monitorización. Más tarde, se explicará en qué consiste, y para que se emplea un framework de programación web, dando además una visión de los más usados actualmente. Finalmente, se mencionarán los dos frameworks GWT[1] y Vaadin[2] que se van a evaluar en este proyecto.

2.1. Aplicaciones Web

2.1.1. ¿Qué es una aplicación web?

Según el diccionario de la Real Academia de la Lengua Española (RAE[3]), la definición de aplicación informática es la siguiente:

> Programa preparado para una utilización específica, como el pago de nóminas, formación de un banco en términos léxicos, etc

En base a la definición anterior, podemos decir que una aplicación web es una herramienta que nos permite desarrollar una tarea específica a través de la web; es decir, a través de un navegador. En el caso de las aplicaciones de escritorio, la carga producida por esas tareas es ejecutada en el propio sistema en que está instalada la aplicación o bien por un servidor web, caracterizándose siempre por contar con un cliente instalado en el sistema operativo que se esté empleando. No es así en el caso de las aplicaciones web, donde la carga de trabajo es trasladada siempre a un servidor web, el cual realiza las operaciones necesarias, llevando las salidas producidas al cliente web o navegador.
A través de una navegador web se accede a dos tipos de elementos: páginas web y aplicaciones web. La diferencia básica entre ellas dos, es que en las primeras la información

[1] http://www.gwtproject.org/
[2] https://vaadin.com/
[3] http://www.rae.es

es estática, al contrario que en las segundos, en la que es dinámica. Esta dinamicidad es la que permite la realización de tareas.

2.1.2. Historia de las aplicaciones web

Las primeras aplicaciones, a las cuales podríamos considerar las precursoras de las aplicaciones web actuales, podrían ser las formadas por las aplicaciones con la arquitectura cliente-servidor. Estas se servían de un programa cliente, el cual debía ser instalado en cada una de las máquinas en las que se quisiera emplear. Habitualmente, un mínimo cambio en el servidor, requería de un cambio en el cliente asociado. Esto suponía tener que estar actualizando constantemente el software, para que fuese compatible con la última versión del servidor, lo que conllevaba un elevado coste de mantenimiento y una productividad que se veía afectada por las actualizaciones.
Ante este panorama, la web era la herramienta que permitiría la gran evolución de las aplicaciones web. La web comenzó siendo una plataforma formada por colecciones de páginas estáticas, documentos y demás información. A continuación se relacionan una serie de acontecimientos sobre la evolución de la web, a los cuales podríamos considerar como aquellos que dieron lugar a las aplicaciones web, tal y como las conocemos hoy en día.

- En el año 1995, Netscape[4] introdujo un lenguaje de scripting para el lado del cliente, llamado JavaScript. Este lenguaje, permite realizar la modificación de ciertos elementos de la página web, así como validaciones, de una forma dinámica. Todo ello en el lado cliente, sin necesidad de realizar invocaciones al servidor para traer constantemente recursos estáticos, tal y como funcionaba la web.

- En el año 1996, Macromedia (en la actualidad Adobe Systems[5]) desarrolló Flash; un reproductor de vectores animado. Esto permitía la inclusión de animaciones en las tradicionales páginas web, con tan sólo la adicción de un plugin.

- En el año 1999, el concepto de aplicación web fue introducido en el lenguaje Java en la especificación del servlet[6], versión 2.2. En este se definía a una aplicación web como:

  > A web application is a collection of servlets, html pages, classes, and other resources that can be bundled and run on multiple containers from multiple vendors.
  >
  > Una aplicación web es una colección de servlets, páginas html, clases y otros recursos, que pueden ser agrupados y ejecutados en múltiples contenedores de múltiples proveedores

- En el año 2005, apareció el término AJAX como una agrupación de técnicas y tecnologías web que permitían que en el lado del cliente, las aplicaciones fueran

[4] https://isp.netscape.com/
[5] http://www.adobe.com/es/
[6] https://jcp.org/aboutJava/communityprocess/final/jsr154

asíncronas. Esto posibilita, que cada vez que se quisiera comunicar con el servidor, no se tuviera que recargar la página entera, minimizando mucho el gasto de ancho de banda y maximizando la experiencia del usuario.

Después de estos acontecimientos, y con el conjunto de las tecnologías detalladas, podemos decir que se llega a las aplicaciones web tal y como se conocen actualmente. La web elimina todas las restricciones nombradas en cuanto a las aplicaciones tradicionales de escritorio, permitiendo así una vía permanente de actualización [10].

2.1.3. Justificación del uso de una aplicación web

Una estrategia actual que están adoptando las empresas de software, es la de proveer acceso a su software a través de aplicaciones web, eliminando así el paso de distribución de aplicaciones locales. Consecuentemente el modelo de negocio está cambiado. En la actualidad, en lugar de adquirir el producto, se abona una cuota periódica y gradual según las funcionalidades que se necesiten de la aplicación. No obstante, muchas de las aplicaciones actuales son gratuitas para el usuario, ya que se financian principalmente a través de la publicidad. A las empresas e instituciones que siguen esta estrategia se las conoce como proveedores de servicios de aplicaciones (application service provider - ASP). El software que proveen es conocido como software de servicio (software as a service - SaaS). Su característica principal es el método de distribución, en el cual el software y los datos se encuentran ubicados en la nube. Se podría decir que las aplicaciones web están de moda por las bondades con las que cuentan [9].
A continuación se muestran una serie de citas, con respecto al mercado actual, que dan una idea del gran impacto que están teniendo estas tecnologías en el mundo actual. Según un informe de la empresa Siemer[7]

> Companies with SaaS models are expected to grow revenue almost 17 % globally this year (2013), despite a sluggish European market.
>
> Se espera que las compañías con estrategia SaaS puedan aumentar sus ingresos casi un 17 % a nivel mundial este año (2013), a pesar de un mercado Europeo lento
>
> The number of SaaS companies receiving capital has been remarkably strong, with growth of 25 % from the previous year
>
> El número de empresas con modelo SaaS que han recibido capital ha sido muy fuerte, con un crecimiento del 25 % respecto al año anterior

Por lo tanto, si la industria ha dado este paso, es porque existen una serie de ventajas que hacen de las aplicaciones web un servicio totalmente viable frente a las aplicaciones tradicionales. Esto se debe principalmente a una serie de argumentos, algunos de los cuales se muestran a continuación:

- No requiere instalación, la aplicación corre en un navegador web.

[7] http://www.siemer.com/news/siemer-associates-releases-summer-2013-saas-industry-report/

- Actualizaciones rápidas y sin que apenas afecten al usuario.
- Viable en cualquier plataforma a través de un navegador web.
- Disponibles desde cualquier lugar del mundo con conexión a Internet, las 24 horas del día, los 7 días de la semana.
- Bajo consumo de recursos en el sistema que lo ejecuta.
- La información está centralizada. Esto permite una mayor seguridad y facilidad para el mantenimiento de backups.

Las aplicaciones también cuentan con ciertos inconvenientes derivados de su propia naturaleza, algunos de ellos son:

- Lento, en caso de que la conexión a internet lo sea.
- Internet no está siempre disponible al 100 %
- No disponibilidad del servicio.
- Las aplicaciones no suelen ser igual de sofisticadas que las aplicaciones de escritorio.
- Riesgos de seguridad derivados de Internet.
- Compatibilidad de los navegadores web.

Después de un breve repaso a las bondades y debilidades de las aplicaciones web, se justifica su elección para el desarrollo de la interfaz de monitorización por los siguientes motivos:

- Independencia del sistema operativo, ya que tan solo es necesario un navegador compatible.
- Es lo más actual. El sistema de monitorización está pensado para monitorizar sistemas cloud, es decir, lo actual. Por lo tanto, desarrollar un cliente de escritorio sería ilógico.
- Existen multitud de frameworks para el desarrollo de aplicaciones web. La mayoría de estos frameworks están orientados a un sencillo y rápido desarrollo, para tener una aplicación medianamente viable, y con aspecto visual aceptable, en muy poco tiempo.

2.1.4. Frameworks para el desarrollo de aplicaciones web

El desarrollo de una aplicación web con un framework de programación o sin él, se podría comparar al montaje de una mesa, con las piezas que la componen previamente fabricadas, o teniendo que fabricarlas antes de montarla. Es decir, un framework de programación de aplicaciones web permite el desarrollo para la web, obviando muchas de las tareas que ésta requiere, como: la gestión de peticiones al servidor, la creación de html, el envío de respuestas al cliente, etc. Además de los detalles relacionados directamente con la web, la mayoría de estos frameworks proporcionan librerías para la abstracción de múltiples tareas. Algunas de ellas son: el acceso a bases de datos, frameworks para templates, manejo de sesiones, etc. Esto permite que el desarrollo se centre directamente en la programación de la propia aplicación, obviando muchas tareas de más bajo nivel. Consecuentemente, aumentando la productividad del desarrollo. Otra característica de los frameworks de programación de aplicaciones web, es que la mayoría de ellos abogan por el desarrollo del código, de manera que éste sea altamente reutilizable.

La reutilización del código tiene mucho que ver con la forma en que se estructuran las aplicaciones que se desarrollan. Una arquitectura bien definida, permite un desarrollo más modular, y por lo tanto reutilizable. Mayoritariamente, los frameworks de programación se basan en el patrón de diseño modelo-vista-controlador (MVC). Se discutirá acerca de este patrón en el capítulo 4 de la memoria. Simplemente señalar que este patrón, propone la separación de la arquitectura de la aplicación en tres elementos: el modelo de datos, la interfaz y la lógica de negocio.
Todos los frameworks de desarrollo de aplicaciones web cuentan con una serie de características que hacen del framework una herramienta muy útil. Todo ello debido a la gran abstracción que proporciona. Algunas de estas características son:

- Sistema de templates web. Consiste en un procesador de templates que hace que sea posible generar páginas web, a partir de dichos templates y cierta información; lo que permite una gran abstracción sobre contenidos en lenguaje HTML.

- Una caché. Ésta se encarga de almacenar cierta información para reducir de una manera considerable la comunicación y el ancho de banda entre el cliente y el servidor.

- API's para el acceso a bases de datos. Permitiendo un acceso de alto nivel. Gestores de URLs. Estos simplifican la tarea de mapear cada una de las urls solicitadas, con el código que debe de ser ejecutado.

- Ajax. Necesario para permitir que la web sea dinámica, sin necesidad de que ésta se esté recargando constantemente, posibilitando además la comunicación cliente-servidor, de una manera transparente para el usuario.

Como se ha visto, estas características permiten a los desarrolladores abstraerse de muchas tareas a la hora de desarrollar aplicaciones. Además, no es necesario un conocimiento total de todas las tecnologías de la web como: HTML, CSS, JS, etc. Finalmente, si

a esto se le une la popularidad de las aplicaciones web, se produce la aparición de multitud de frameworks, para la mayoría de los lenguajes de programación más conocidos [8]. A continuación se muestra el cuadro 2.1 con el número de frameworks de programación más usados por o para cada uno de los lenguajes a su vez más populares.

Lenguaje	**N de Frameworks**
ASP.NET	6
C	1
C++	4
Java	32
JavaScript	8
Scala	3
Perl	5
PHP	26
Python	18
Ruby	6

Cuadro 2.1: Nº de Frameworks más usados por cada lenguaje de programación [15]

Como se puede ver en la tabla, existen en total unos 100 frameworks de programación de aplicaciones web para los distintos lenguajes de programación. Dar una visión general de cada uno de ellos no sería abordable. Por lo tanto, debido a la popularidad en todos los ámbitos del lenguaje Java, y a su mayor número de frameworks, se hará un breve estudio sobre algunos de los más usados en dicho lenguaje, tratando así de explicar el por qué de tan gran número de frameworks. Las diferencias, aunque existentes, no se van a centrar en el número de características del framework, como por ejemplo si tiene gestor de URLs o no, sino en diferencias en cuanto a su complejidad, documentación, etc.

A continuación se muestran los frameworks de programación de aplicaciones web en Java más usados. Primero, se darán datos básicos sobre ellos, como es la fecha de aparición y su última versión. En el cuadro 2.2 se proporciona dicha información. Después se mostrarán datos de su uso. Finalmente se analizarán diferentes aspectos como: facilidad de uso, calidad de las interfaces, complejidad, etc; para poder así realizar una comparativa. Como se puede comprobar, dado que es una tecnología actual, los diferentes frameworks no presentan grandes diferencias en cuanto a sus fechas de creación. Se podría decir que los primeros aparecieron a partir de la definición en el año 1999 del término aplicación web.

Nombre	Primera versión	Versión de la comparativa	Última Versión
Spring MVC	Junio 2013 - 1.0 -	- 3.2.3 -	Marzo 2014 - 4.0.3 -
Struts	Mayo 2000 - 1.0 -	- 2.3.15.1 -	Octubre 2006 - 2.3.-16.1 -
GWT	Marzo 2006 - 1.0 -	- 2.5.0 -	Enero 2014 - 2.6.0 -
JSF	Marzo 2004 - 1.0 -	- 2.2 -	Enero 2014 - 2.2.5 -
Vaadin	Mayo 2009 - 6.0 -	- 7.1.1 -	Diciembre 2013 - 7.1.9 -
Grails	Julio 2005 - 0.1 -	- 2.2.2 -	Abril 2014 - 2.3.8 -
Wicket	Junio 2005 - 1.0 -	- 6.8 -	Febrero 2014 - 6.14.0 -
Play!	Mayo 2008 - 1.0 -	- 2.1.2 -	Marzo 2014 - 2.2.2 -

Cuadro 2.2: Versiones de la comparativa sobre los frameworks más usados en Java [12]

Ahora se van a mostrar datos de su uso. Según un estudio de la empresa ZEROTURNAROUND[8], la repartición es la que se muestra en la figura 2.1.

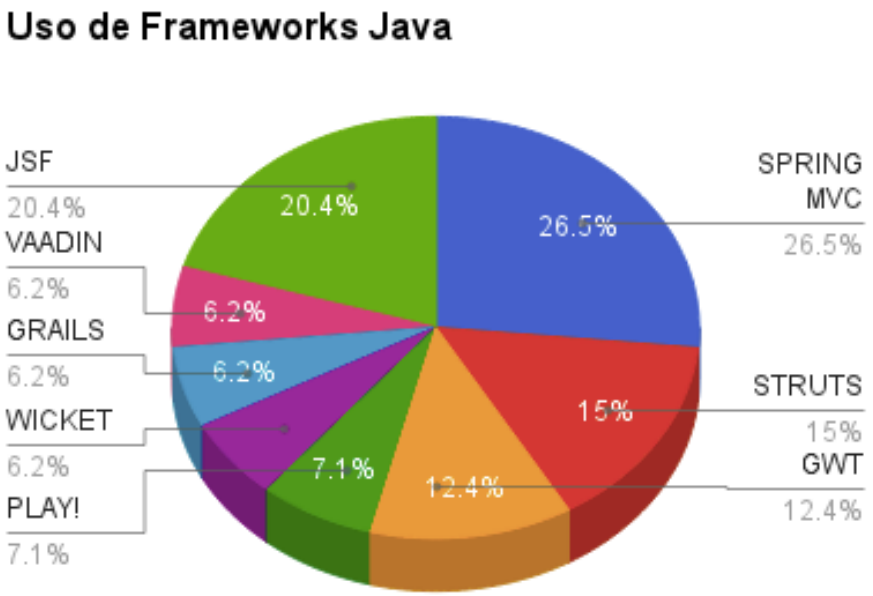

Figura 2.1: Porcentaje de uso de los frameworks más usados en Java [12]

[8]http://zeroturnaround.com/

Se podría decir que su uso está bastante repartido, a pesar de haber 3 frameworks que acaparan más del 50 % del mismo. También se debe tener en cuenta que en estos datos se puede ver repercutido el tiempo de vigencia de cada framework, propiciando un uso mayor de los más antiguos, lo cual nos indica la mayor o menor calidad de unos u otros. Por lo tanto se va a dar una visión en base a 8 aspectos generales, para tratar de mostrar las bondades y carencias de cada uno de los frameworks siguientes: JSF[9],Vaadin[10],Grails[11],Wicket[12],Play![13],Spring MVC[14], Struts[15] y GWT[16]. Con el objetivo de ver qué framework se adapta mejor al tipo de desarrollador y a sus necesidades, los 8 aspectos a valorar son los siguientes:

Rápido prototipado de las aplicaciones Este apartado se centra en la rapidez con la que el framework ayuda a realizar un rápido prototipo de una aplicación. Es decir, la cantidad y complejidad del código será la menor posible. En este apartado Grails y Play! son los dos frameworks más adecuados. El primero cuenta con un setup muy rápido y con generación de código, la que ahorra mucho tiempo. El segundo se caracteriza por una estructura inicial de las aplicaciones, que permite un despliegue inmediato. A tener en cuenta también el framework Vaadin. Éste cuenta con muchos módulos fácilmente integrables, junto a un modo de edición de visual que genera el código automáticamente, consiguiendo unos layouts con muy buenos resultados.

Complejidad del framework Habitualmente, cuantas más funcionalidades presenta el framework, mayor complejidad provoca a la hora de su desarrollo. Consecuentemente un mayor tiempo para entender su arquitectura y una menor productividad. Por lo tanto, hay que buscar la equivalencia entre más funcionalidad y menos complejidad. En este apartado Vaadin y GWT son los dos frameworks más adecuados, perseguidos por Struts. Decir que Vaadin se basa en GWT, por lo que la simplicidad que ofrece GWT, es la misma en Vaadin. Esta simplicidad reside en que tan solo hay que preocuparse de seleccionar los componentes que se quieren mostrar y en la lógica mediante código Java. Finalmente este código se compila a JavaScript de una manera muy eficiente, lo que permite su despliegue en la mayoría de los navegadores.

Facilidad de uso A pesar de estar relacionado con los dos anteriores, se podría definir como el tiempo que transcurre desde que se descubre el framework, hasta que se obtienen los primeros resultados. En este apartado destacan Grails and Vaadin debido a la facilidad para su configuración. El primero se caracteriza por una adicción de plugins muy simple, sin configuración alguna, tan solo por un comando. El segundo cuenta con un modo de edición de layouts visual, y que genera el código de una manera totalmente transparente,

[9]http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html
[10]https://vaadin.com/
[11]https://grails.org/
[12]http://wicket.apache.org/
[13]http://www.playframework.com/
[14]http://spring.io/
[15]http://struts.apache.org/
[16]http://www.gwtproject.org/

permitiendo crear componentes que a su vez pueden ser insertados dentro de otros, sin una sola línea de código. Además, el código generado tiene una estructura muy sencilla, y muy fácil de ser modificada, para añadir cambios no permitidos desde el editor visual.

Documentación y comunidad Importante contar con ambos elementos. Habitualmente se podría decir que una gran comunidad, aparte de fomentándose, se crea con una buena documentación. Esta documentación permitirá formas a una comunidad consistente, incluso de cara al propio futuro desarrollo del framework. En este caso Grails y Vaadin son los clarísimos ganadores. Ambos frameworks cuentan con infinidad de información documentando ambas herramientas. Desde wiki, foros, guías; hasta videos tutoriales subidos por los propios desarrolladores. Además estos mismos desarrolladores están muy activos en los foros, lo cual es de agradecer.

Entorno del framework Es decir, la cantidad de funcionalidades extra que el framework presenta. Estas pueden ser integración con bases de datos, inclusión de plugins, etc. Habitualmente un mayor número de funcionalidades puede llevar a una mayor complejidad. Grails y Play! cuentan con muchas soluciones de integración, las cuales permiten ampliar enormemente las capacidades del framework. Grails en concreto cuenta con plugins para gestores de dependencias como Maven[17] y Ant[18], plugins de acceso a base de datos noSQL como MongoDB[19] y Neo4j[20], etc.

Rendimiento y estabilidad Este punto es muy importante a la hora de decidir el alcance de la aplicación a desarrollar. Es decir, no es lo mismo una aplicación para el uso de unas cuantas personas que una aplicación para miles, manteniendo un rendimiento adecuado y razonable. En este apartado Play! es el claro ganador. Le siguen Vaadin y GWT. Play permite una gran escalabilidad junto a un buen rendimiento, gracias a su integración con Akka actors[21]. Akka permite la creación de aplicaciones concurrentes y distribuidas en la máquina virtual de Java (JVM). Vaadin y GWT presentan un muy buen rendimiento, debido al eficiente código que el compilador de GWT genera sobre el lenguaje JavaScript, para ser ejecutado en el navegador.

Mantenimiento del código y actualizaciones A tener en cuenta si la planificación de la aplicación es a largo plazo. Esto es, si en cada actualización se añaden nuevas funcionalidades, estas deben de ser integradas a la perfección con la estructura anterior, no provocar cambios en el desarrollo. Grails es el claro ganador en este apartado. Esta desarrollado siguiendo el paradigma de diseño software de convención sobre configuración (convention over configuration), lo que permite minimizar el número de decisiones de diseño del desarrollador, lo que maximiza la simplicidad, pero sin necesidad de perder flexibilidad.

[17] http://maven.apache.org/
[18] http://ant.apache.org/
[19] http://www.mongodb.org/
[20] http://www.neo4j.org/
[21] http://akka.io/

Interfaces Este es uno de los puntos más importantes, ya que al final, una aplicación web tiene que ver mucho con lo que el usuario espera. Si la creación de las interfaces es una tarea muy compleja, esto provocará que el resultado visual final, sea el de una aplicación muy pobre, o sino un gran esfuerzo de desarrollo a cambio. Vaadin y GWT son dos frameworks muy completos en cuanto a simplicidad del desarrollo y resultado visual final. Ambos cuentan con multitud de componentes prefabricados como gráficas, formularios, botones, etc; los cuales tan solo deben ser añadidos. Vaadin además cuenta con una opción de temas configurables, lo que permite que una vez hayas desarrollado tu aplicación, los temas y aspectos sean cambiados con una sola línea de código, seleccionando alguno de los muchos estilos predefinidos de los que dispone. GWT cuenta con un sistema similar pero un poco más limitado.

Se ha realizado una breve revisión de los puntos fuertes y de las debilidades, de ocho frameworks de programación en Java. Esto ha servido para dar una visión de lo que existe actualmente en el mercado, y por lo tanto, tratar de reconocer qué herramientas se adaptan mejor a nuestras necesidades para el desarrollo del proyecto. Se han seleccionado los frameworks Vaadin y GWT para un análisis más profundo en el siguiente capítulo, y así tratar de decidir finalmente con que framework se desarrollará la aplicación web del proyecto. La elección de estos dos frameworks se ha basado principalmente en tres puntos: el primero la similitud entre el modelo de programación de los dos frameworks, con lo cual, lo que se aprende en el estudio de uno, sirve para el estudio de otro; el segundo la facilidad de uso y la amplia documentación del framework, lo que permite un rápido estudio y el desarrollo de una aplicación funcional, en un tiempo razonable, de acuerdo a la situación; el tercero por todos los módulos de componentes visuales que vienen prefabricados en ambos frameworks, lo que permite un desarrollo visual en un tiempo mucho menor con respecto al resto, y con un resultado más que aceptable y profesional [12].

Capítulo 3

FRAMEWORKS ESTUDIADOS

En este capítulo se va a describir todo lo relacionado con los dos frameworks (GWT[1] y Vaadin[2]) propuestos para el desarrollo del proyecto. En la subsección 2.1.4 del anterior capítulo, ya se realizó una introducción sobre qué es un framework para el desarrollo de aplicaciones web. Por lo tanto, en la primera sección de este capítulo se describirá de manera breve el estudio realizado sobre los dos frameworks propuestos: se dará una introducción a ellos, se describirán los ejemplos realizados, y finalmente, se justificará la elección de uno de ellos. En la sección final se dará una visión de las principales características que ofrece el framework seleccionado para el desarrollo, Vaadin.

3.1. Trabajos previos con GWT y Vaadin

GWT y Vaadin son los dos frameworks de programación de aplicaciones web sobre los que se llevó a cabo un estudio. El objetivo de éste era el de tratar de decidir cual de los dos frameworks se adaptaba mejor a las necesidades para el desarrollo de la aplicación web, para el sistema de monitorización. El estudio consistió en realizar algunos tutoriales sobre el desarrollo con ambos frameworks, que sirvieron para familiarizarse con estos, y ver cual de ellos ofrecía un mejor resultado con respecto al tiempo invertido en el desarrollo. A continuación se realiza una breve descripción de cada framework, y se muestra un ejemplo realizado con cada uno de ellos.

Tanto en GWT como en Vaadin se desarrolla en lenguaje Java. Para las partes de la aplicación que deben de ejecutarse en el navegador, este código Java es traducido a código JavaScript. GWT es software libre ofrecido por Google[3] bajo una licencia Apache[4], mientras que Vaadin es un producto comercial de una empresa con ese mismo nombre.

GWT Es un grupo de herramientas de software libre que permiten crear complejas aplicaciones web en Java, compilarlas a JavaScript, y ejecutarlas en un navegador web. Las

[1] http://www.gwtproject.org/
[2] https://vaadin.com/
[3] http://www.google.com/about/company/
[4] http://www.apache.org/licenses/

aplicaciones están soportadas por la mayoría de los navegadores web existentes. Estas aplicaciones son ejecutadas totalmente en la parte cliente (navegador), y tan solo acceden a servidores para almacenar o leer información. GWT por defecto cuenta con una serie de componentes visuales como: botones, tables, paneles, gráficas, etc; que pueden ser usados para la creación de la interfaz de la aplicación.
Para probar todas estas características se ha realizado alguno de los tutoriales[5] que se ofrecen en la página de GWT [2]. En estos se han podido crear una serie de sub-aplicaciones para probar las distintas posibilidades que ofrece el framework. Entre éstas, se ha realizado una tabla de estadísticas con los valores de las acciones de determinadas compañías, pero sobre todo, casos relacionados con gráficas. En la figura 3.1 se puede ver el resultado de una aplicación que se conecta mediante RPC a un servidor, del que obtiene valores, y los muestra en la gráfica, ordenados por el tiempo actual.

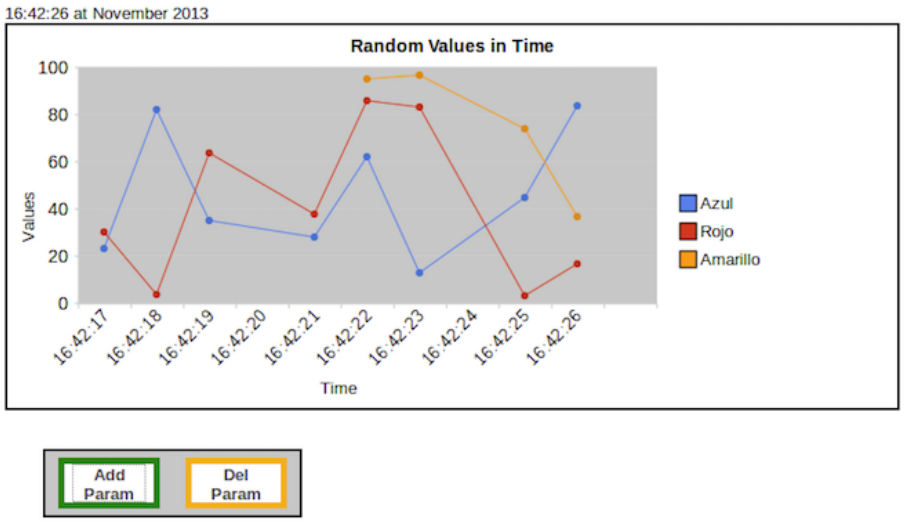

Figura 3.1: Primera gráfica desarrollada con GWT

Vaadin Está formado por: una API para la programación del lado servidor, una API para el lado cliente, un conjunto de componentes y widgets para ambas partes, temas para el control de la apariencia, y un modelo de datos que permite enlazar directamente componentes del lado servidor con fuentes de información. Para el lado cliente, se cuenta con un compilador que permite traducir el código Java a JavaScript. Decir que este compilador es una extensión del de GWT. Vaadin permite tanto el desarrollo de aplicaciones solo a nivel de cliente, como GWT, así como el desarrollo en la parte servidora. Señalar que a pesar de ofrecer ambas posibilidades, está mas orientado a esta última, presentando unas mejores características. Al igual que con GWT, con Vaadin se realizaron varios ejemplos de aplicaciones. Sobre todo se tomaron ideas de las páginas[6] [7] de demos de Vaadin, y se trató de modificar el código que ahí se proporcionaba. Decir que el desarrollo es muy

[5]http://www.gwtproject.org/doc/latest/tutorial/index.html
[6]http://demo.vaadin.com/book-examples/book/
[7]http://demo.vaadin.com/charts/

similiar al de GWT. La mayoría de los ejemplos se centraron en la parte de gráficas, con los que en muy poco tiempo se obtuvieron resultados como los de la figura 3.2.

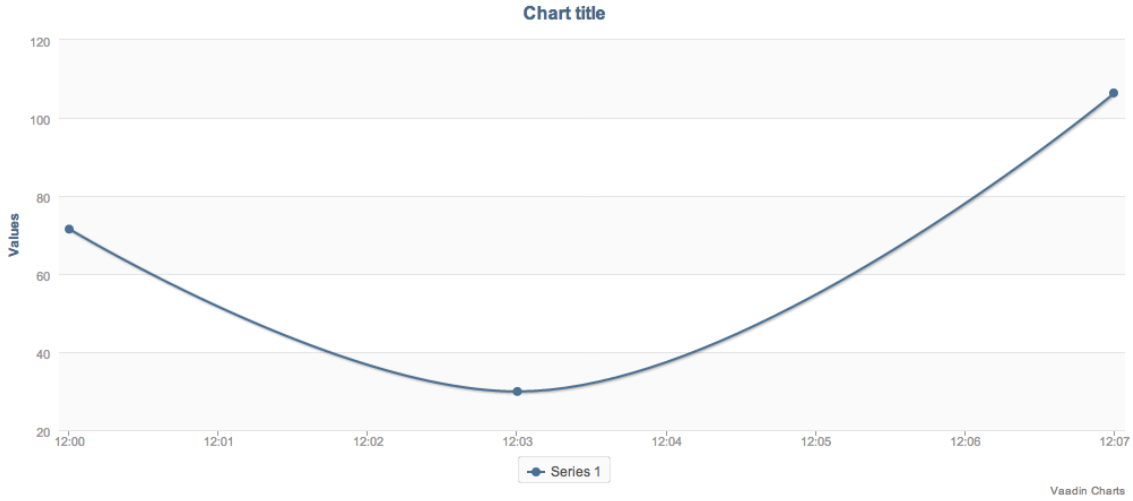

Figura 3.2: Primera gráfica desarrollada con Vaadin

Como conclusión general, se podría decir que GWT es más adecuado para aplicaciones en las que su ejecución está casi limitada al navegador, es decir, que tiene que conectarse muy pocas veces a un servidor para leer o escribir información. Por ejemplo un videojuego, una aplicación de creación de diagramas, etc. Mientras que en Vaadin, las aplicaciones se ejecutan mayoritariamente del lado del servidor, y en la parte del cliente tan solo se muestra la vista y se capturan eventos de ésta. Esto permite que el usuario no tenga acceso a la lógica de la aplicación, lo que en algunos casos puede ser necesario por temas de seguridad. El punto en contra de Vaadin, es que la comunicación permanente entre el navegador y el servidor, puede repercutir en el rendimiento para conexiones con poco ancho de banda. Con respecto a las herramientas que se ofrecen y la facilidad para desarrollar con estas, Vaadin es el claro vencedor. Este cuenta con una serie de componentes visuales prefabricados, que permiten crear vistas con una calidad considerablemente mejor que las realizadas con GWT, en el mismo o mayor tiempo. En las dos imágenes de las figuras 3.1 y 3.2 se puede ver una comparativa entre más o menos el mismo tiempo de desarrollo, y el resultado final, el cual es claramente mucho más aceptable en Vaadin.
Por lo tanto, el framework que se ha elegido para el desarrollo es Vaadin, principalmente por dos motivos:

- Desarrollo y ejecución principal en la parte servidora, lo que incrementa notablemente la seguridad del sistema.
- Cuenta con un mayor grupo de componentes, y de más calidad; además de permitir una simple y amplia configuración, tanto visual como lógica, la cual en GWT es mucho más costosa. Especial mención a la librería de gráficas, que será una de las piezas claves de la aplicación del sistema de monitorización.

3.2. Introducción a Vaadin

En esta sección se va a ofrecer una visión general acerca de los temas mas importantes a la hora de desarrollar una aplicación con el framework Vaadin. Decir que la visión que se va a proporcionar va a ser de alto nivel, no entrando en detalles de clases, interfaces o métodos. También a tener en cuenta que para el desarrollo con Vaadin, tan solo es necesario tener conocimientos de programación en Java, y en caso de que se quieran modificar los estilos visuales por defecto de Vaadin, conocimientos de HTML y CSS. A continuación se muestra una lista de los temas que se van a tratar a lo largo de esta sección:

- **Arquitectura**. Explicación de la arquitectura de Vaadin.
- **Desarrollo en la parte servidora**. Elementos a tener en cuenta para el desarrollo en la parte servidora.
- **Componentes de la interfaz**. Descripción de todos los componentes disponibles.
- **Paneles y layouts**. Descripción de los layouts disponibles.
- **Temas**. Temas predeterminados de Vaadin y como modificarlos.
- **Manejo de Información**. Relación entre los componentes y la información.
- **Gráficas**. Descripción de todos los tipos de componentes gráfica.

Arquitectura

Vaadin cuenta con dos modelos para el desarrollo de aplicaciones: desarrollo en la parte del cliente (navegador web), o desarrollo en la parte servidora. El primero de ellos permite el desarrollo de aplicaciones en Java, las cuales son compiladas en JavaScript y ejecutadas en el navegador. El segundo modelo se desarrolla totalmente en la parte servidora y en Java. Este ejecuta se ejecuta principalmente en la parte servidora, y la parte visual es compilada a JavaScript y renderizada para ser mostrada en el navegador. En ambos modelos se comparten componentes visuales (botones, etiquetas, etc), temas y código de back-end. Es decir, con el desarrollo del lado cliente nos encontramos con una aplicación que se ejecuta directamente en el navegador, mientras que con el desarrollo en la parte servidora, la ejecución se realiza en la parte servidora, solo que siendo mostrada en la parte cliente. La comunicación en el segundo modelo entre la parte servidora y cliente, se realiza a través de la tecnología AJAX.
En la figura 3.3 se puede ver una ilustración de la comunicación entre el lado cliente y el lado servidor. Las aplicaciones desarrolladas para la parte servidora, ejecutan como servlets[8] sobre un servidor web Java, y responden a respuestas HTTP. El servlet se encarga de recibir las peticiones del lado cliente, y traducirlas a eventos que sean conocidos por el lado servidor de la aplicación. En muchas ocasiones esta petición requiere ser respondida, en este caso, el servlet envía una respuesta a la parte cliente, el cual la recibe, y en caso de que sea necesario, realiza cambios en la interfaz.

[8]http://www.oracle.com/technetwork/java/overview-137084.html

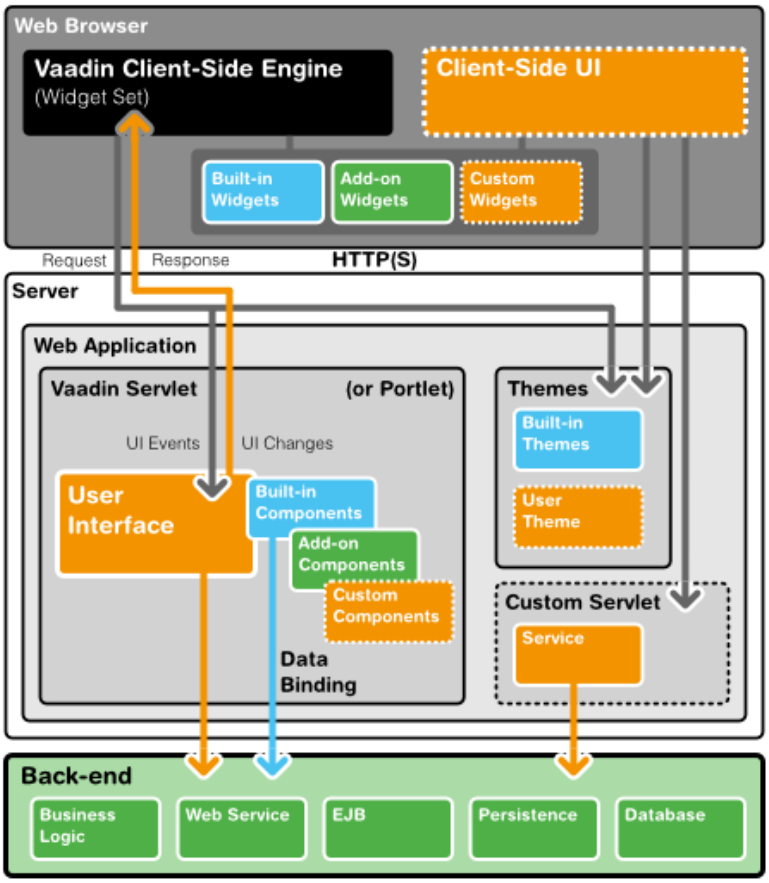

Figura 3.3: Ilustración de la comunicación entre el lado cliente y servidor [4]

Vaadin está compuesto por diferentes tecnologías que componen su arquitectura. Estas son: HTML y JavaScript, CSS y Sass, AJAX, GWT y Servlets de Java.

HTML y JavaScript HTML se emplea para definir la estructura y el formato de las páginas web, y permitir la inclusión de imágenes y otros recursos. Vaadin emplea la versión 5 de HTML, la cual es soportada por la mayoría de los navegadores. JavaScript se emplea para poder incluír programas en las páginas HTML, y hacerlas dinámicas. Además de para capturar los eventos de la interfaz y comunicarlos.

CSS y Sass Cascading Style Sheet (CSS) es un lenguaje para definir la parte visual de la arquitectura definida por HTML. Esto es: colores, tamaños de los textos, etc. El encargado del emparejamiento entre HTML y CSS es el propio navegador. Syntactically Awesome Stylesheets (Sass) es una extensión del lenguaje CSS. Este permite el uso de variables, herencias, etc; lo cual hace más llevadero la definición de estilos. En Vaadin se pueden emplear ambos lenguajes, aunque finalmente todo termina por compilarse a CSS que es lo que es capaz de interpretar el navegador.

AJAX Asynchronous JavaScript and XML (AJAX) es una técnica empleada en el desarrollo de aplicaciones web con interación del usuario. Este permite que la parte cliente se comunique con la parte servidora, sin necesidad de recargar la página nuevamente.

GWT La parte cliente de Vaadin está basada en el Google Web Toolkit. En Vaadin el desarrollo se realiza en código Java, el cual debe de ser traducido al lenguaje JavaScript para poder ser ejecutado en el navegador. Esto se consigue a través del compilador de Vaadin, que es una extensión del compilador de GWT.

Java Servlets Un Servlet de Java es una clase que es ejecutada en un servidor web Java, la cual permite extender las funcionalidades del servidor. Ésta es la encargada de recibir las peticiones HTTP de la parte cliente, y la que genera las respuestas, las cuales pueden estar formadas por HTML, para modificar el contenido actual.

Desarrollo en la parte servidora

El desarrollo de la parte servidora consiste en la creación y el control de los componentes que forman la interfaz de usuario. Estos componentes son alimentados con información, y modificados visualmente a partir de los estilos. Para este desarrollo se cuenta con una serie de elementos que permiten la creación de la aplicaición web. A continuación se describe a cada uno de ellos:

UI Las aplicaciones en Vaadin se comienzan a desarrollar a partir de esta clase inicial UI. Ésta es el punto de conexión entre la sesión de un servlet y la aplicación. Es decir, por cada instancia de la aplicación en un navegador se genera una nueva UI, a partir de la cual se extiende toda la aplicación en el servidor.

Page Cada UI está asociada a un Page. Un objeto Page representa a la página web, así como a la ventana del navegador en donde la UI se ejecuta. Este objeto sirve para una vez dentro de la aplicación, acceder a información sobre el tamaño de la ventana, ver si está en segundo o en primer plano, etc.

Sesión de Vaadin Es un objeto que representa a una sesión del usuario asociada a una UI. Esta sesión se inicia cuando el usuario abre una instancia de la aplicación en el navegador, y se cierra cuando la sesión expira en el servidor, o el usuario la cierra explícitamente.

Componentes y layouts de la interfaz de usuario Estos componen una jerarquía de componentes que es la que forma la interfaz de usuario. El usuario interacciona con estos componentes, provocando eventos en relación a dicho componente, que son capturados por la aplicación. Los componentes van desde botones, pasando por etiquetas, hasta gráficas. Además se pueden crear componentes desde cero o extender los ya existentes.

Eventos y listeners Vaadin sigue un patrón básico, en el cual la interacción en la aplicación se traduce en eventos y listeners que reciben dichos eventos. Este es el modelo entre el cliente y el servidor. El cliente genera eventos y el servidor los recibe. También es posible establecer un patrón en el que el servidor tiene la capacidad de generar eventos, invirtiendo así el flujo tradicional. Esto es muy útil para tareas periódicas de actualización de la parte cliente por parte de la servidora.

Recursos En las aplicaciones de Vaadin se pueden incluír ciertos recursos como es el caso de las imágenes. Estos recursos son almacenados en la parte servidora.

Temas En Vaadin, la lógica y la vista de la interfaz de usuario están separadas. Esto permite un desacoplamiento que es muy adecuado para la reutilización del código, y un desarrollo independiente. La lógica es gestionada a través del lenguaje Java, y la vista a través del lenguaje CSS. Vaadin por defecto cuenta con una serie de estilos predefinidos para sus componentes, que pueden ser aplicados fácilmente.

Modelo de datos Vaadin permite enlazar sus componentes que requieren información, directamente a fuentes de datos. Esto lo consigue a través de una serie de conectores adaptados a distintas tecnologías, que permiten la actualización de la información sin ningún control.

Componentes de la interfaz

Vaadin cuenta con un amplio abanico de componentes que hacen posible la creación de interfaces muy ricas visualmente. Estos componentes, a los que se les denomina finales, siempre están contenidos dentro de layouts o paneles, que los posicionan dentro de la vista de la aplicación. Además de los componentes predeterminados, Vaadin ofrece la posibilidad de heredar de estos y modificarlos, o de crear nuevos desde cero. Decir que estos componentes son compartidos por el desarrollo, tanto a nivel de servidor, como cliente. Muchos de estos componentes tienen asociados contenedores de información de los que se hablará más adelante. A continuación se van a listar algunos de los más usados. Para consultar la lista completa de éstos o visualizarlos, dirigirse al libro de Vaadin referenciado en la bibliografía [4].

- **Etiqueta**. Sirve para mostrar texto no modificable.
- **TextField**. Permite la entrada de información textual.
- **Botón**. Empleado para iniciar determinadas acciones por parte del usuario.
- **CheckBox**. Es un componente de dos estados, que permite ser modificado por el usuario, y leído por la aplicación.
- **Tabla**. Permite mostrar información organizada por filas y columnas.
- **Árbol**. Se emplea para representar de una manera natural información que tiene relaciones jerárquicas.

Paneles y Layouts

Los panales o layouts, junto a los componentes finales descritos en el punto anterior, forman la interfaz de las aplicaciones en Vaadin. Los paneles y layouts se emplean para posicionar a los componentes finales, en determinados lugares de la interfaz. Decir que la interfaz de la aplicación es una jerarquía de layouts y paneles, los cuales tienen en sus hojas a componentes finales. Como es de imaginar, un layout puede contener en su interior a otros layouts, y por supuesto a componentes finales. El posicionamiento de los paneles, la definición de sus tamaños, alineación de estos, etc; se realiza todo a nivel de código. Para evitar esta tarea lenta y tediosa, Vaadin proporciona un plugin para eclipse[9] con el cual se puede realizar esta tarea de manera visual e interactiva. Se puede ver una imagen de dicho editor en la figura 4.29.
Los principales paneles y layouts que ofrece Vaadin se detallan a continuación:

- **Layout Vertical**. Permite posicionar otros layouts o componentes finales en orden vertical. Habitualmente tiene tamaño fijo a lo ancho e indefinido a lo largo.
- **Layout Horizontal**. Permite posicionar otros layouts o componentes finales en orden horizontal. Habitualmente tiene tamaño fijo a lo largo e indefinido a lo ancho.
- **Layout Absoluto**. Permite posicionar otros layouts o componentes finales en una posición arbitraria dentro del panel. Siempre tiene tamaño fijo tanto a lo largo como a lo ancho.
- **Panel**. Permite posicionar otros layouts o componentes finales tanto en orden vertical como horizontal. Además, al contrario que con los layout vertical y horizontal, en caso de que los componentes de su interior sean de mayor tamaño que él, despliega un scroll en horizontal, en vertical o en ambos sentidos; para poder visualizar así todo el contenido de su interior.
- **Sub-Ventana**. Es un layout secundario que se superpone al layout principal de navegador. Permite posicionar dentro de él a otros layout o componentes finales.
- **Split Horizontal y Vertical**. Permite posicionar otros layouts o componentes finales en orden vertical u horizontal. La característica que lo diferencia es que divide su área en dos, mediante una barra intermedia que puede ser desplazada en vertical o horizontal según el tipo de layout. Esto permite ajustar el tamaño de las dos áreas según la necesidad.

Temas

Los temas en Vaadin permiten tener el control sobre la apariencia visual de las aplicaciones. Estos temas se crean usando el lenguaje Saas, el cual es una extensión del lenguaje CSS, o directamente a través de CSS. Vaadin por defecto cuenta con una serie de temas que están especificados en Saas, y son posteriormente compilados a CSS. Estos temas son:

[9] http://www.eclipse.org/

- Valo
- Reindeer
- Chamaleon
- Runo
- Liferay

Para el desarrollo de una aplicación se puede seleccionar uno de los temas por defecto, extender uno de ellos y modificarlo, o crear uno nuevo desde cero. Se recomienda extender alguno de los ya existentes y realizar variaciones en caso de que se quiera un estilo propio. La tarea de crear un estilo a partir de cero solo es viable para aplicaciones muy simples. Vaadin está diseñado de manera que permite la independencia entre el desarrollo de la lógica y los temas. Esto posibilita que se pueda cambiar la apariencia de una aplicación, tan solo con modificar una linea, en la que se le especifica el tema por defecto.

Manejo de Información

Vaadin cuenta con un modelo de datos, que permite enlazar los contenedores de información de los componentes finales, con fuentes de información como bases de datos. Esto facilita que a la hora de desarrollar se elimine mucho código de control. Por defecto, Vaadin cuenta con múltiples conectores, para realizar el enlace de los contenedores de los componentes con determinadas tecnologías, principalmente de bases de datos.
Comos se ha visto, de una parte está el modelo de datos de Vaadin y por otro lado las fuentes de información. Con respecto al primero, el framework proporciona un modelo formado por tres elementos relacionados entre sí. Se pueden ver estos en la figura 3.4.

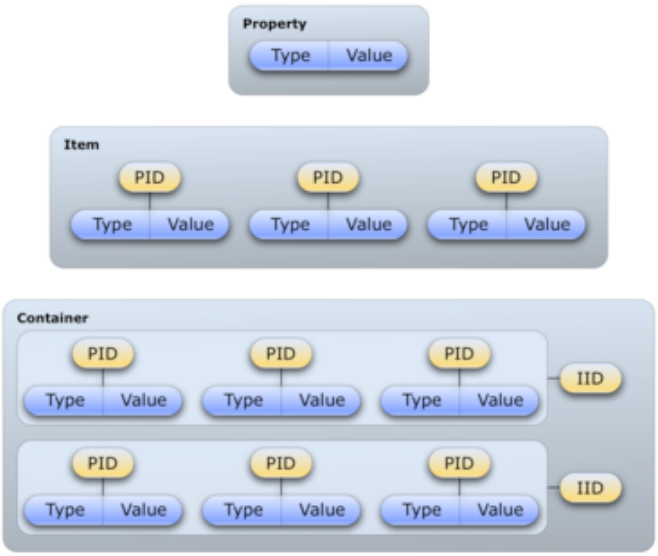

Figura 3.4: Modelo de datos en Vaadin [4]

Este modelo permite la relación entre los componentes finales de la interfaz y los datos. Se podría hacer una analogía por ejemplo con las tablas, que estarían representadas por el container; las filas, que estarían representadas por el item; y las celdas, que estarían representadas por la property. Esta analogía puede ser extendida al resto de componentes en Vaadin. La edición del contenido de los elementos del modelo, es lo que permite que los componentes varíen la información que estos muestran en la interfaz. Es decir, para cambiar el valor de una celda de una tabla, es necesario modificar el valor dentro del objeto property, asociado al container de la tabla.

Gráficas

Las gráficas en Vaadin se pueden considerar como una especie de componentes finales especiales. Estos están contenidos en una librería que no se incluye por defecto en el paquete de Vaadin. Esta librería proporciona una serie de gráficas muy ricas visualmente, con las que el usuario además puede interaccionar. Estas gráficas permiten la visualización de datos numéricos en una o dos dimensiones, además de su configuración visual tanto de forma como de estilos. Las gráficas se pueden dividir en dos tipos: gráficas normales y timeline. Las primeras son las más abundantes, las segundas tan solo cuentan con un tipo de vista para la visualización de grandes cantidades de datos en series de tiempos. Dentro de las primeras se encuentran muchos tipos de gráficas para visualizar la información de distintas maneras. A continuación se muestran en la figura 3.5 algunas de las más básicas.

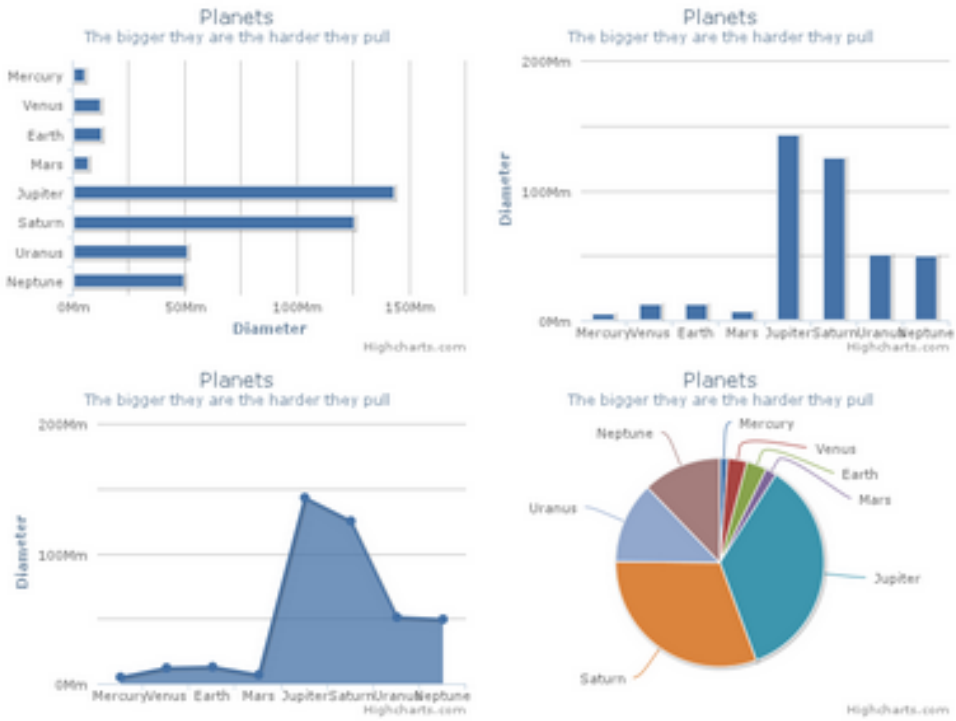

Figura 3.5: Ejemplo del paquete de Gráficas en Vaadin [4]

Como todos los componentes, las gráficas tienen asociado un objeto modelo en el cual se recoge la información que se quiere mostrar. Además, los componentes gráficas, tienen otro objeto asociado en el que se especifica toda la configuración visual. Esta va desde mostrar o no el título en la cabecera de la gráfica, a la definición del tamaño de los puntos que en ésta se visualizan [4].

Capítulo 4

ARQUITECTURA DE LA APLICACIÓN WEB

En este capítulo se van a discutir todos los temas relacionados con la arquitectura de la aplicación web de monitorización, tratando así de dar una idea y una visión que permitan su perfecta comprensión. Para comenzar, se realiza una introducción al patrón MVC y como ha sido implementado en la aplicación. A continuación se describe el funcionamiento del sistema de monitorización, y como éste mantiene la consistencia. Más tarde se hablará del diseño de la aplicación, y como es el funcionamiento de ésta. Finalmente se explicará todo lo relacionado con las vistas y estilos de la interfaz de la aplicación web.

4.1. Patrón Model-View-Controller (MVC)

En este apartado se va a a realizar una breve introducción histórica al patrón MVC, junto con la motivación de su aparición. A continuación se explicará el mismo. Más tarde se mostrará la visión de este patrón que se recomienda para el desarrollo con el framework de Vaadin (MVP). Finalmente, se explicará de qué manera se ha interpretado este patrón en el desarrollo de la aplicación web.

4.1.1. MVC

MVC es un patrón de arquitectura de aplicaciones software. Su aparición y motivación se podría resumir, según palabras de su creador Trygve Reenskaug [1], de la siguiente manera:

> I made the first implementation and wrote the original MVC reports while I was a visiting scientist at Xerox Palo Alto Research Laboratory (PARC) in 1978/79. MVC was created as an obvious solution to the general problem of giving users control over their information as seen from multiple perspectives
>
> Hice la primera implementación, y escribí la primera descripción del patrón MVC mientras estaba visitando como científico el laboratorio de investigación de Xerox

[1] http://heim.ifi.uio.no/ trygver/

en Palo Alto (PARC) en 1978/1979. El patrón MVC se creó como una solución al problema general de dar el control al usuario sobre la información, y que ésta se pueda mostrar desde diferentes perspectivas

Trygve Reenskaug realizó una primera aproximación al patrón introduciendo los conceptos de THING-MODEL-VIEW-EDITOR. Él mismo, junto a otras personas, evolucionaron estos conceptos hacia los actuales MODEL-VIEW-CONTROLLER. Éstos se pueden definir de una manera breve de la siguiente forma:

- **Modelo**. Representa el conocimiento, la lógica de negocio, la información a mostrar.
- **Vista**. Es la representación visual del modelo.
- **Controlador**. Es el enlace entre el modelo y la vista.

Este patrón ha tenido mucho éxito a lo largo de los años, llegando hasta la actualidad, debido en gran parte a la gran evolución de la web, y también porque introduce dos conceptos muy útiles como son: la reutilización de código y la separación de conceptos, que facilitan su desarrollo y posterior mantenimiento [13].

El patrón de diseño permite separar muy bien los conceptos de model-view-controller a la hora de desarrollar el código. Esto posibilita que cada uno de estos conceptos se puedan desarrollar por separado, permitiendo así una mayor flexibilidad e independencia, a la hora de realizar cambios o de reutilizar las implementaciones. Cada uno de los conceptos del patrón tienen una clara funcionalidad.

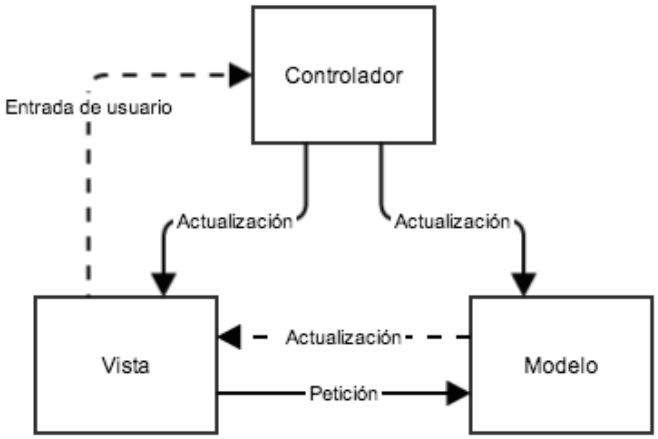

Figura 4.1: Diagrama del Patrón MVC.

- **El controlador** es el encargado de responder a los eventos que el usuario genera en la vista de la aplicación. Este debe de gestionar dichos eventos, y realizar los cambios necesarios en el modelo, para actualizarlo. Además, es el encargado de modificar la vista, en cuanto a la manera en que ésta se presenta al usuario. No confundir con los datos que presenta.

- **El modelo** es la información y la lógica de negocio que se emplea en la aplicación. El modelo puede ser desde la información de una base de datos, a las normas a aplicar sobre las entradas del usuario en la aplicación. El modelo se activa por peticiones del controlador, y él mismo se encarga de actualizar la vista con la información que se haya solicitado.

- **La vista** es la representación visual para el usuario del modelo, además del lugar en el que éste interacciona con la aplicación. La vista envía eventos al controlador y esta es actualizada por el modelo.

Como se ha visto, la interacción entre los diferentes componentes del patrón se podría definir como una interacción circular. La vista envía un evento al controlador, éste actualiza el modelo, y es este modelo el que vuelve a actualizar a la vista. Obviamente el patrón y sus pequeñas evoluciones presentan una mayor complejidad que la mostrada, pero esta explicación sirve para hacerse una idea general del patrón [11].
Como se ha dicho, el patrón ha sufrido pequeñas modificaciones, en muchos casos derivadas de la tecnología sobre la que se implementa el patrón. Estas modificaciones pueden ser conceptuales, o relacionadas con la manera de implementar el patrón en el desarrollo software.
El desarrollo para la web, tanto de páginas como aplicaciones web, ha tenido una fuerte acogida del patrón MVC. En muchos casos, la interpretación del patrón se ha visto modificada para adecuarse a la tecnología a emplear. Por ejemplo, la repartición de responsabilidades sobre los conceptos del patrón no está muy clara entre el cliente y el servidor.

4.1.2. Vaadin - MVP

A continuación se va a dar una visión de cómo el framework Vaadin, usado en el desarrollo, propone la interpretación y uso del patrón MVC. Vaadin emplea el patrón Model-View-Presenter (MVP), que es una variación conceptual del patrón original MVC. Se va a describir en qué consiste y en qué varía con respecto al patrón MVC, pero desde una perspectiva más cercana a la implementación y desarrollo, que hacia una visión conceptual.
El patrón MVP surge para tratar de mejorar la lógica de presentación. Este patrón mantiene los tres elementos del MVC, salvo que el controlador se pasa a llamar presentador (presenter). Este cambio es consecuencia de una modificación en las funciones a desarrollar por los diferentes componentes de la arquitectura.
Como se puede observar, se podría considerar al presentador como el director de orquesta, en detrimento de la vista y del modelo que pierden funcionalidades. A la vista se la podría considerar pasiva, ya que esta tan solo se encarga de mostrar la parte visual, sin apenas contar con lógica para el manejo de la interacción del usuario, que es relegada totalmente al presentador. Es el presentador el que se encarga de procesar dichos eventos, obtener la información del modelo, y finalmente actualizar la vista. Además de actualizar la información de la vista, el presentador es el encargado de procesar la información y determinar como debe de ser mostrada en la vista. Se podría considerar al presentador como un intermediario entre la vista y el modelo.

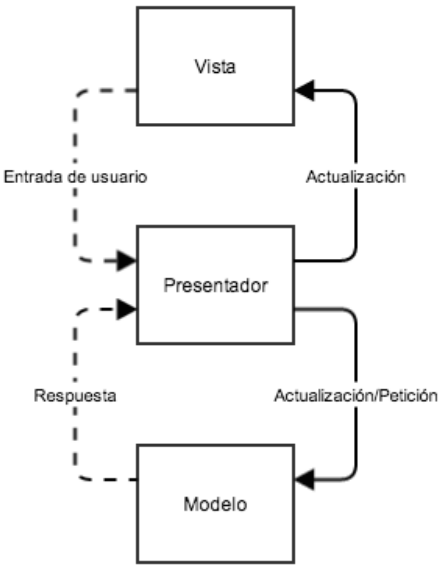

Figura 4.2: Diagrama del Patrón MVP.

Este cambio del patrón hacia una arquitectura de intermediario contrasta con el del patrón MVC, en donde el flujo se podría considerar circular. El beneficio de este cambio principalmente es la clara separación y dependencia entre la vista y el modelo. Esto permite delegar las relaciones tan sólo en el presentador, lo que facilita también la realización de pruebas en el sistema que se desarrolle.

Como se ha mencionado, Vaadin propone el uso del patrón MVP, como base para el desarrollo de aplicaciones con un cierto tamaño. A grandes rasgos, se podrían comparar los componentes del patrón con los componentes de la aplicación web. Es decir, se podría realizar la equivalencia entre cliente-vista, servidor-presentador e información-modelo; como se ha visto en la introducción a Vaadin en el capítulo anterior.
Desde una perspectiva más cercana al desarrollo, Vaadin propone aislar totalmente al presentador. Esto se consigue a través de la definición de interfaces entre el presentador y la vista; y entre el presentador y el modelo, para que así el presentador pueda interactuar con la vista y el modelo, a través de una interfaz, permitiendo la separación del desarrollo de las diferentes partes. También se propone un modelo de eventos o llamadas asíncronas, para que tanto la vista como el modelo, le puedan comunicar eventos al presentador cuando lo necesiten.

En la figura 4.3 se muestra como propone Vaadin la implementación de una aplicación. Se debe de tener en cuenta que el caso que se muestra es un caso simple, con una sola instancia de vista, una instancia de presentador y una instancia de modelo. El desarrollo de una aplicación con un mínimo tamaño, es en la realidad mucho más complejo. De todas formas, el diagrama sirve perfectamente para ilustrar la modelización del patrón a la hora del desarrollo [11] y [3].

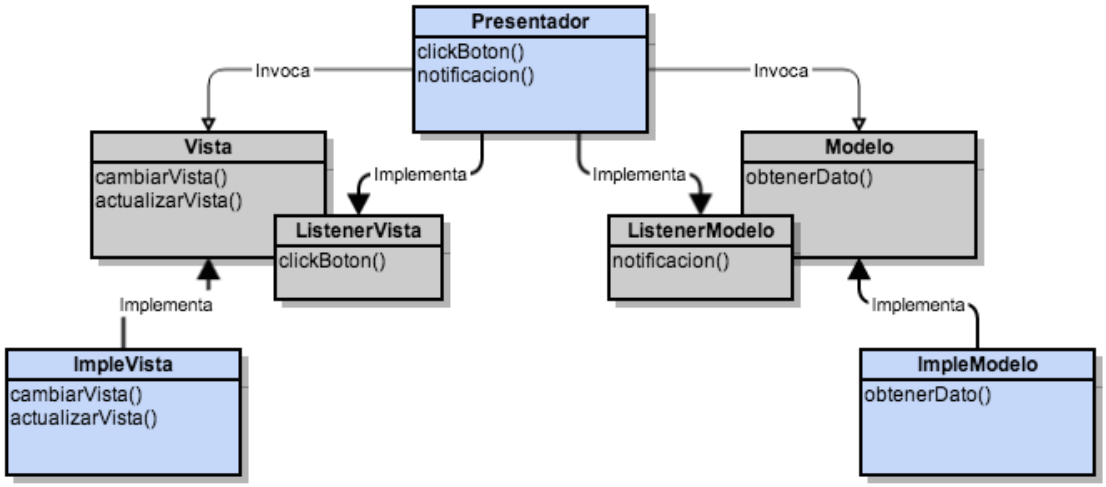

Figura 4.3: Diagrama de componentes para implementar el Patrón MVP.

4.1.3. Patrón empleado

En el desarrollo de la aplicación, se ha optado por el uso del patrón MVP como recomienda Vaadin. Este patrón puede tener distintas variaciones en cuanto a su implementación se refiere. Para el desarrollo se ha optado por modificar un poco la estructura propuesta por Vaadin, con respecto a la interacción de los componentes. En parte como consecuencia de que el modelo teórico, visto anteriormente, es perfectamente válido para una aplicación simple. Sin embargo para la aplicación que se va a desarrollar, debe de ser modificado, o así se ha considerado. A continuación se van a mostrar dos diagramas, figuras 4.4 y 4.5, en los que se pretende representar la interacción de los componentes, y reflejar la interpretación del patrón MVP que se ha seguido en la implementación de la aplicación.

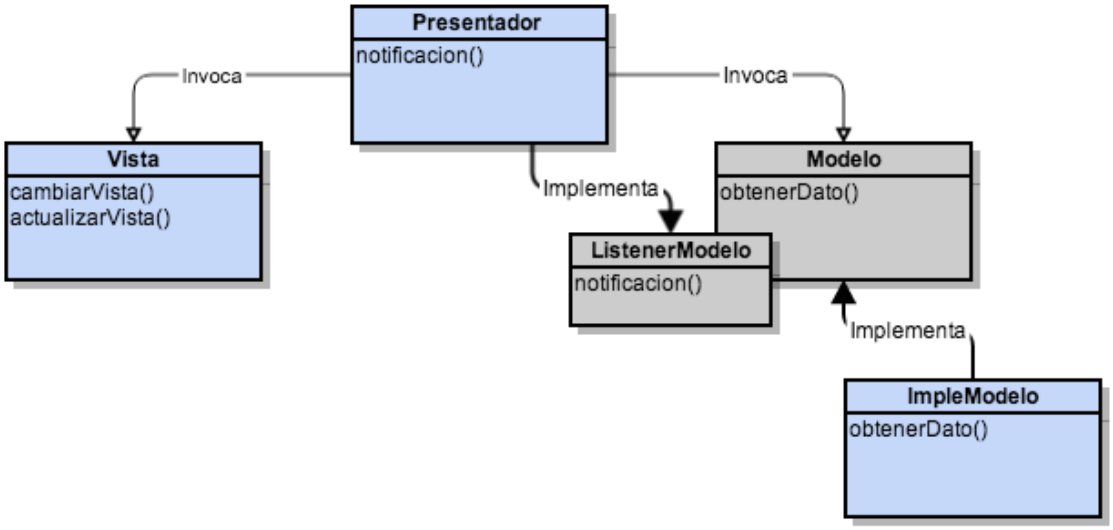

Figura 4.4: Diagrama del patrón seguido para la implementación del modelo 1.

La figura 4.4 muestra la interacción de los componentes para el caso en el que una parte del modelo, necesita notificar eventos al presentador. Se aclarará en la sección 4.2 quien conforma el modelo de la aplicación. Básicamente, el presentador sigue siendo el que se encarga de prácticamente toda la lógica de la aplicación, como se proponía en el patrón inicial.
La vista en este caso es un componente totalmente pasivo, tan solo se encarga de soportar la parte visual, y ésta es actualizada directamente por el presentador. Además, es el propio presentador el encargado de implementar las interfaces de los eventos que el usuario genera en la vista, sin que esta necesite manejar ni notificar nada al presentador.
El modelo de datos es consultado por el presentador. Además, este modelo de datos cuenta con la capacidad de comunicar al presentador eventos que sean de su interés.

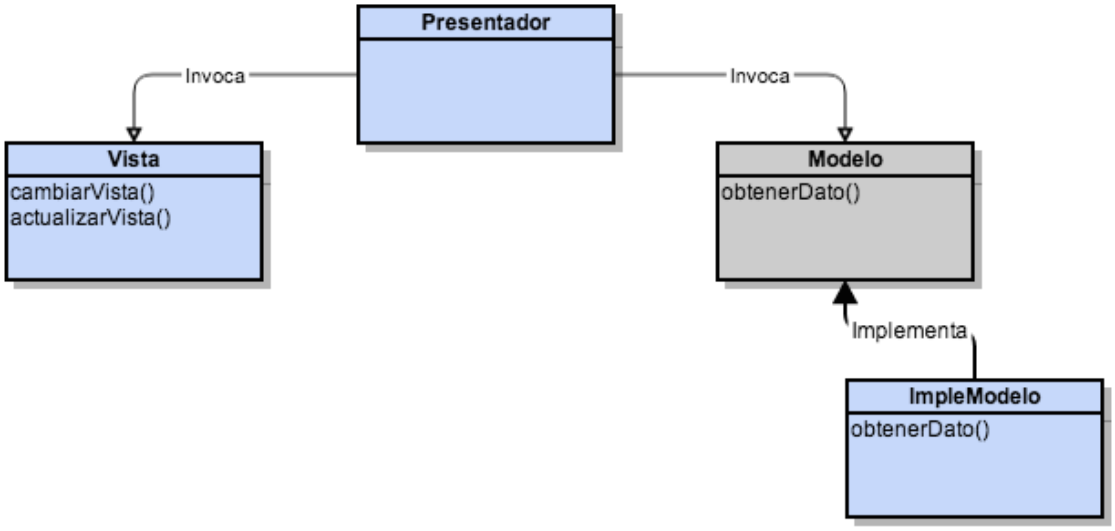

Figura 4.5: Diagrama del patrón seguido para la implementación del modelo 2.

La figura 4.5 muestra la interacción de los componentes para el caso en el que una parte del modelo no necesita notificar eventos al presentador. El resto del funcionamiento es el mismo que para el caso anterior.
Con las dos figuras anteriores se ha explicado la interpretación del patrón MVP que se ha realizado en el proyecto. Las variaciones con respecto al patrón propuesto por Vaadin y los motivos de estas se describen a continuación:

- Se ha suprimido la definición de listeners en las vistas, debido principalmente a que se ha considerado a la vista como un conjunto de componentes visuales sin capacidad para almacenar ninguna lógica. En parte debido a que el diseño de las interfaces se realiza desde una herramienta que autogenera el código necesario para la vista que se diseñe. Si a este código se le añaden elementos no relacionados solamente con la parte visual, esta herramienta no es capaz de visualizar las vistas, lo que es un impedimento a la hora de realizar cambios. Por lo tanto, se ha considerado a la vista totalmente inerte en cuanto a lógica. Esto se ha resuelto implementado los listeners de los eventos de los componentes que conforman las vistas (botones, paneles, etc), en el presentador encargado de dicha vista.

- Se han suprimido las interfaces que definían a las vistas. El motivo de esta decisión ha sido principalmente el mismo que en el punto anterior. Además, la definición de sus interfaces para administrar todos estos componentes, sería una tarea laboriosa, al estar las vistas formadas por múltiples componentes.

Esto cambios van en detrimento de la separación del código de los distintos componentes, generando así dependencia entre los elementos que conforman las vistas y los presentadores que las controlan, no obstante, es la única solución viable que se ha encontrado para la realización del desarrollo.

De manera general, se va a definir que partes de la aplicación son las que desarrollan cada uno de los roles del patrón MVP propuesto. También se describirán las tareas de las que se encargan cada uno de ellos.

- Por un lado tenemos las vistas. Como se ha explicado en el capítulo 3 sobre el funcionamiento de Vaadin, esta se podría comparar a la parte cliente de la aplicación. Éstas componen a las vistas que se muestran en el navegador para la interacción del usuario, además de recibir a través de los componentes que las forman la interacción del usuario, para que ésta sea gestionada por el presentador correspondiente.
- El modelo está formado por la información que alimenta a la aplicación. Esta se obtiene a partir de una base de datos distribuida y de un registro altamente disponible. Estas fuentes de información se explicarán en la sección 4.2.
- Los presentadores son los encargados de toda la gestión de la comunicación entre la vista y el propio presentador; además de entre el presentador y el modelo. Al presentador se le podría comparar con el servidor en cuanto a la arquitectura de la aplicación.

4.2. Separación de cometidos del sistema de monitorización

En esta sección se van a explicar los diferentes cometidos que desarrolla cada uno de los componentes del sistema de monitorización. Primero se describirá de una manera breve, y sin entrar en detalles de implementación, el funcionamiento general del sistema de monitorización y su flujo de información. A continuación se detallarán los mecanismos mediante los cuales el sistema de monitorización mantiene la consistencia.

4.2.1. Descripción

Un sistema de monitorización es un sistema capaz de obtener información del entorno donde se despliega, para su posterior análisis. El objetivo del sistema sobre el que trata este trabajo es la recolección de métricas del sistema a monitorizar, el almacenamiento de éstas, y su visualización. Los valores de dichas métricas permitirán controlar el sistema y llevar a cabo las acciones adecuadas para la gestión y administración del sistema. En este caso, el sistema a desarrollar llevará a cabo tres tareas básicas para la monitorización:

- Recolección de las métricas de cada uno de los componentes del sistema total que se seleccionen.
- Recepción de dichas métricas enviadas por los recolectores, y almacenamiento de éstas para su consulta posterior.
- Visualización a través de una interfaz de todo el sistema, alimentado por las métricas que se recolectan.

El sistema de monitorización se caracteriza por permitir que cualquier sistema, independientemente de su arquitectura, pueda ser monitorizado mediante una visualización que mantenga su arquitectura real junto a sus relaciones. Además, de que por cada nivel de dicha arquitectura, se presenten métricas del sistema que permitan su monitorización. El sistema cuenta además con las siguientes características:

- Es capaz de monitorizar cualquier sistema que se adapte a la configuración.
- El sistema a monitorizar puede tener un número arbitrario de niveles lógicos.
- Es capaz de monitorizar tanto componentes lógicos como físicos.
- Modelo de almacenamiento fiable y con persistencia, lo que permite almacenar de una manera segura toda la información que se recopila de la monitorización.
- Modelo de almacenamiento distribuido para evitar cuellos de botella.
- Los componentes que lo forman están desacoplados.

Todas estas características se pueden entender mejor viendo la figura 4.6 que ilustra la arquitectura formada por todos los componentes del sistema.

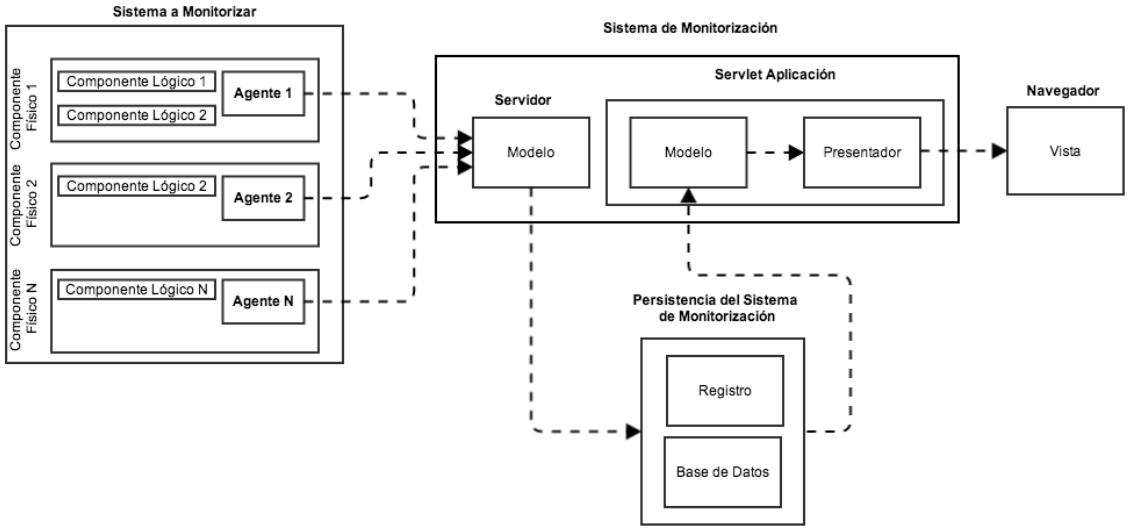

Figura 4.6: Componentes del sistema de monitorización

A partir de la figura 4.6 se va a explicar desde una perspectiva de alto nivel el funcionamiento del sistema. Como se puede ver en la imagen, el sistema se compone de 3 partes claramente diferenciadas:

- **Agentes**. La parte de recopilación de datos del sistema a monitorizar.
- **Servidor y persistencia del sistema de monitorización**. La parte de almacenamiento de dichos datos.
- **Servlet aplicaciones y navegador**. La parte de visualización de los datos.

Como se ha mencionado anteriormente, estas son las tres tareas básicas y fundamentales que realiza el sistema. En la figura 4.6, las flechas indican el principal flujo de la información recopilada en la monitorización. Para que la información que se recopila sea la misma que se solicita visualizar en la interfaz, se requiere una consistencia en todo el sistema de monitorización. El funcionamiento de esto se explicará en la subsección 4.2.2. El proceso de recolección de la información para la monitorización, comienza cuando ésta es exportada por los componentes que componen el sistema que se va a monitorizar. Tanto los componentes lógicos como físicos. Los agentes son los encargados de recolectar esta información y enviarla al servidor. Es este servidor el que se encarga de almacenarla en la base de datos distribuida. Una vez la información se encuentra almacenada, esta es consultada según se solicite a través de la aplicación web de monitorización.
A continuación se va a desglosar la figura 4.6 en subfiguras, para un estudio con mayor detalle, de cada una de las tareas realizadas en cada paso de la monitorización.

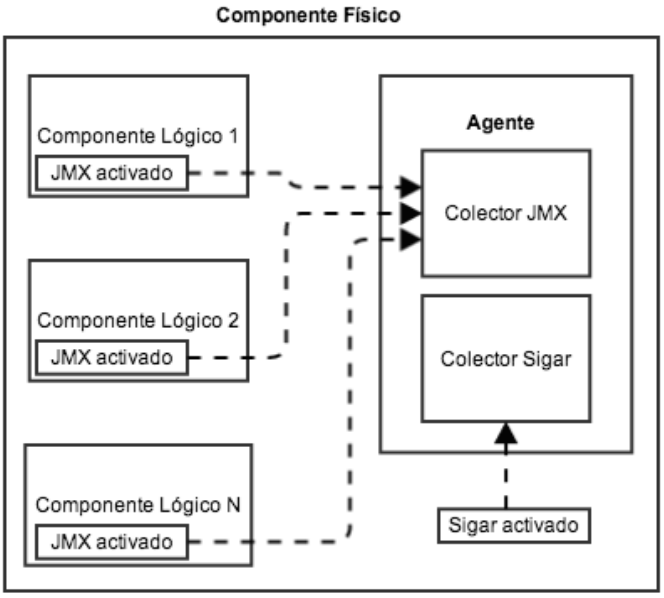

Figura 4.7: Diagrama de la recolección de métricas por parte del agente

Como se ha mencionado en el capítulo 1, el sistema se encarga de la recolección de métricas, tanto de los componentes físicos como de los lógicos, que forman el sistema a monitorizar. En la figura 4.7 se puede ver como un agente se encuentra ejecutándose en un nodo físico. Este agente mediante dos tipos de colectores y mediante polling, se encarga de la recopilación de las métricas del sistema físico sobre el que se está ejecutando; junto con las métricas que exportan las distintas instancias de los componentes de la jerarquía lógica. Las métricas del componente físico se recogen a través de un colector de Sigar[2]. A su vez, las métricas de las instancias de los componentes lógicos son exportadas a través de JMX[3]. A medida que el agente va recopilando las métricas que se exportan, este las envía al servidor. El envío se produce con un cierto retraso con respecto a la realidad. Además, éstas se envían haciendo push al servidor.
Algo a tener en cuenta, es que los componentes físico y lógicos a monitorizar, exportan una serie de métricas, de acuerdo a como han sido programados. Los agentes del sistema de monitorización, están diseñados para que la recolección solo se produzca de las métricas que se hayan especificado en la configuración del sistema. Como ya se ha dicho, esta configuración se especificará en la subsección 4.2.2.
Una vez que las métricas son enviadas al servidor por uno o varios agentes, estas son almacenadas para su posterior consulta.

[2]https://support.hyperic.com/display/SIGAR/Home
[3]http://www.oracle.com/technetwork/java/javase/tech/javamanagement-140525.html

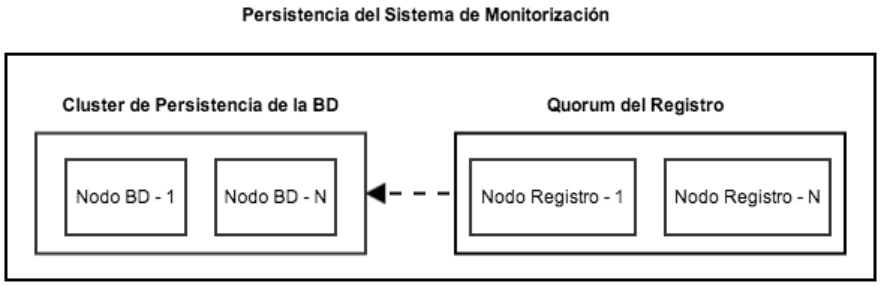

Figura 4.8: Diagrama del almacenamiento del sistema de monitorización

Como se puede ver en la figura 4.8, el almacenamiento del sistema se realiza sobre una base de datos distribuida, lo que proporciona persistencia y evita cuellos de botella. A tener en cuenta que cuando el sistema esté en ejecución, en la misma base de datos se están almacenando las métricas por parte del servidor y realizándose consultas por parte de la aplicación web.
En la figura también se puede observar al registro. Éste es un registro de alta disponibilidad que proporciona la consistencia al sistema, para que de alguna manera, la aplicación web se entienda con la parte de recolección y almacenamiento, sin que exista comunicación entre ellas. Su principal funcionalidad es la de almacenar información crucial para el funcionamiento y consistencia del sistema, estando ésta siempre disponible.

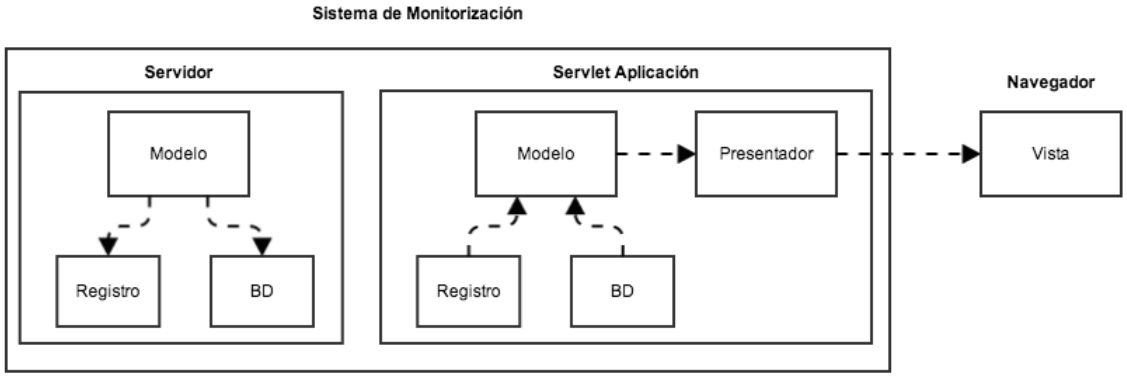

Figura 4.9: Diagrama del servidor y la aplicación web

En esta figura 4.9 se diferencian dos de los principales componentes del sistema. El servidor cuenta con dos instancias con un conector a cada una; una para la base de datos y otra para el registro de alta disponibilidad. A través del conector a la base de datos, el servidor se encarga de almacenar las métricas que recibe de los agentes. Por otro lado, el conector con el registro es empleado por el servidor para dos funciones. La primera de ellas es para obtener una imagen del sistema a monitorizar y conocer sobre qué métricas hay que recolectar la información. La segunda es la actualización de este registro en caso de que algunos de los componentes, tanto físicos como lógicos, que se están monitorizando, no se encuentren disponibles.
La aplicación de monitorización se divide en dos partes. Por una lado está el modelo y el

presentador o controlador, que corren en la parte servidora dentro de un servlet de aplicaciones. Por otro lado está la vista que ejecuta dentro del navegador del usuario.
Al igual que el servidor, la aplicación cuenta en su modelo con dos instancias de dos conectores, uno para la base de datos y otro para el registro. El conector a la base de datos, sirve a la aplicación para solicitar las métricas de los componentes que el usuario quiere monitorizar. El conector al registro sirve para obtener una imagen del sistema a monitorizar, además de información que servirá a la aplicación para saber de qué manera debe de consultar las métricas a la base de datos.

4.2.2. Consistencia

En la subsección 4.2.1 se ha visto el flujo de información del sistema de monitorización, además de los diferentes elementos que los componen. En esta subsección se va a tratar de explicar cómo el sistema mantiene la consistencia, y cual es el punto común sobre el cual el sistema mantiene la lógica. Es decir, tanto el agente que recolecta métricas, como la aplicación que las muestra, deben de tener una visión general del sistema, para que la métrica que se solicita visualizar, sea recolectada del mismo componente que el usuario solicita en la interfaz, ya que como se ha visto, el único punto de encuentro entre la recolección y la aplicación web, es la base de datos distribuida y el registro de alta disponibilidad.
Esto se consigue a través de un fichero XML de configuración, del que se hablará en el capítulo 5. En este fichero se realiza la configuración del sistema de acuerdo al sistema real que se quiera monitorizar. Los puntos más importantes de este fichero se detallan a continuación:

- Jerarquía de componentes físicos y lógicos que definen al sistema a monitorizar.
- Direcciones IP de los componentes físicos donde están ejecutándose los componentes lógicos. Además de la asignación de dichas IPs para cada uno de los componentes lógicos, según la máquina física en la que se ejecutan.
- Referencia mediante un nombre y una versión, a los ficheros de definición de métricas asociados a cada uno de los componentes lógicos y físicos.

Con estos tres puntos principales, además de otros, el sistema a monitorizar queda definido. Principalmente la jerarquía de los componentes define la arquitectura del sistema a monitorizar, la cual es necesaria para la representación visual del sistema. Las direcciones IP especifican las máquinas físicas sobre las que se ejecutan los componentes lógicos. Esto sirve para conocer qué componentes físicos se deben monitorizar, y qué componentes lógicos se deben de monitorizar en cada componente físico. Finalmente, las referencias a los ficheros de definición de métricas sirven para especificarle a cada uno de los agentes recolectores, qué métricas se deben de recolectar de cada uno de los componentes.
Con toda esta información, está lista la configuración inicial para el inicio del sistema. Pero puede ocurrir una cosa, que el sistema comience a monitorizar, y en un determinado momento alguno de los componentes del sistema a monitorizar no se encuentre disponible. Para dar soporte a esta posibilidad, una posible solución sería un fichero dinámico

que se estuviera continuamente actualizando y leyendo. Como esto sería totalmente ineficaz, y por lo tanto no viable, la solución es la de un registro de alta disponibilidad. Este registro lo que permite es tener la información siempre disponible y actualizada.
Por lo tanto, toda la información del fichero de configuración es introducida en el registro de alta disponibilidad, para que desde ahí pueda ser consultada y actualizada por alguno de los dos componentes del sistema de monitorización, que en este caso son el servidor y la aplicación.

Como se ha dicho, en el fichero de configuración está definida la jerarquía que define al sistema a monitorizar. Dado que el registro es una especie de árbol de directorios en el que por cada path o nodo se puede almacenar una entrada de información, nos sirve para almacenar de una manera muy eficiente y realista la información del sistema a monitorizar.
Por lo tanto, si recordamos los elementos esenciales del fichero de configuración: jerarquía, IPs y ficheros; y tenemos en cuenta que la jerarquía es propiamente la esencia del registro; solo falta por almacenar en la entrada de información de cada nodo la IP; junto al nombre y la versión que especifican el fichero de definición de métricas.
Ahora que ya se conocen los elementos que dan consistencia al sistema, se va a explicar el uso de esta información para permitir que a partir del relleno del fichero, la interfaz termine mostrando una métrica.
El proceso comienza con el relleno del fichero de configuración XML con el despliegue del sistema que se va a monitorizar. Obviamente, esta información debe ir acorde con el despliegue real que se vaya a hacer o esté hecho del sistema a monitorizar. El siguiente paso es rellenar el registro con la información del fichero de configuración. Para ello se ha desarrollado un programa (ConfigurationManager) que valida el fichero de configuración y lo parsea. A continuación crea un conector al registro y rellena todo el árbol con la configuración especificada. Una vez el registro cuenta con toda la información de configuración, el servidor lee toda la información del registro. El propio servidor necesita la información para conocer qué ficheros de especificación de métricas se han seleccionado, y por lo tanto, qué métricas se le van a decir a los agentes que deben de recolectar. Además, son necesarias las IPs de cada nodo, para saber a que agente hay que especificarle qué métricas y de que componente deben de ser recolectadas. Es decir, para identificar un componente a monitorizar es necesario el nombre y la clave del fichero de métricas, junto a la IP de la máquina en la que se encuentra. Ahora es el turno de la aplicación web de monitorización. Esta nada más iniciarse procede a leer del registro toda la información que contiene, dado que debe darle al usuario una imagen visual del sistema que se está a monitorizar. Para ello, tan sólo sería necesaria la información que proporcionan los paths del registro con los nombres de los componentes que forman la jerarquía. Pero solo esta información no es suficiente para que la aplicación web funcione. La aplicación necesita conocer los nombres y las versiones de los ficheros de definición de métricas, junto a las IPs, para poder así realizar las correspondientes consultas a la base de datos, que previamente el servidor ha rellenado y organizado según un esquema, a través de su conector.
Finalmente, el papel del registro es el de mantener actualizada en todo momento la configuración del sistema a monitorizar. Esto es, en caso de que un componente físico o lógico del sistema a monitorizar no esté disponible, el servidor debe de notificar al registro pa-

ra que este se actualice y genere los eventos correspondientes. Básicamente, cuando el servidor detecta que alguno de sus agentes se ha caído completamente (nodo físico no disponible), o que no está recolectando métricas de un componente lógico (nodo lógico no disponible), debe de notificarlo al registro. Consecuentemente, el registro envía un evento al monitor para que este actualice la representación de la jerarquía del sistema a monitorizar, y representa que cierto recurso no se encuentra disponible.

Nota: de los temas tratados en este apartado, destacar que son autoría propia todo lo relacionado con la aplicación web y la definición del fichero de configuración, junto a la clase que lo traslada al registro, todo de la parte de consistencia.

4.3. Diseño y funcionamiento de la aplicación

En esta sección se va a realizar una explicación de varios apartados relacionados con la aplicación web. Para comenzar se hará una introducción a las funcionalidades que implementa la aplicación web. A continuación se mostrará el diseño de ésta desde una perspectiva del patrón MVC, haciendo referencia a sus componentes y a sus relaciones. Finalmente se mostrarán los eventos que se producen a lo largo de toda la aplicación, y se explicará el flujo de funcionamiento de la aplicación web, dentro de todo el sistema de monitorización.

4.3.1. Introducción a la aplicación web

La aplicación web tiene como funcionalidad básica la de ser el canal de visualización de las métricas del sistema que se está a monitorizar. Ésta cuenta con una interfaz, que se despliega en un navegador web, y que proporciona una serie de funcionalidades. La primera de ellas es la opción de configurar desde la propia interfaz, el sistema que se quiere monitorizar. Tan solo con indicarle el lugar en el que se encuentra el registro, y la base de datos del sistema a monitorizar, es suficiente para iniciar la monitorización. Una vez que la aplicación se ha iniciado con una configuración, la segunda funcionalidad es la exploración de los componentes lógicos y físicos del sistema a monitorizar. A través de un árbol que representa visualmente el sistema a monitorizar, se puede ir recorriendo su jerarquía y visualizando las propiedades y métricas de sus componentes. Desde el propio explorador, se puede seleccionar una métrica de un componente y visualizarla de manera gráfica. La tercera funcionalidad es el dashboard. Este permite la agregación de múltiples métricas, y que todas ellas sean visualizadas al mismo tiempo de manera gráfica. La adicción de métricas al dashboard se hace desde el explorador a medida que se recorren los diferentes componentes. Mencionar también que cada una de las métricas que se visualizan en forma de gráfica, tienen la opción de ser configuradas para definir la velocidad de refresco, y el número de valores mostrados. Un ejemplo real del uso de la aplicación se puede ver en el capítulo 5.

4.3.2. Diseño jerárquico

Como se ha visto en la sección 4.1 del patrón MVC, la aplicación está formada por tres componentes principales. Estos son la vista, el presentador o controlador, y el modelo. Éste último, se puede considerar que está representado por la información de la base de datos y el registro, junto a los conectores que dan acceso a ellos. Por lo tanto, la aplicación web necesita de controladores y vistas que la formen. A partir de este punto, se llamará controlador a lo que se ha visto en la descripción del patrón como presentador. A grandes rasgos, la aplicación web está formada por clases que representan a vistas y a controladores. La implementación y diseño de las vistas se discutirá en la sección 4.4.

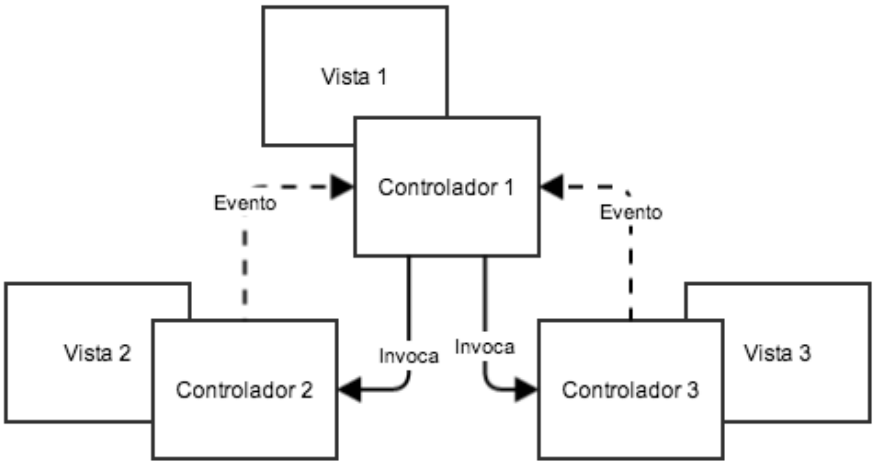

Figura 4.10: Diagrama breve de la arquitectura de la aplicación

Como se puede ver en la figura 4.10, la aplicación es como una especie de jerarquía de vistas gestionadas por controladores. Recordar que como se propuso en el patrón de diseño, las vistas son totalmente inertes, es decir, tan solo se encargan de representar la parte visual sin gestionar ningún tipo de lógica. Esta es tarea de los controladores. Las vistas de mayor nivel contienen en su interior a las de menor nivel. Por su parte, los controladores se encargan de manejar a cada vista de la que son responsables, a procesar eventos de los controladores de las vistas de menor nivel y realizar operaciones sobre dichos controladores.

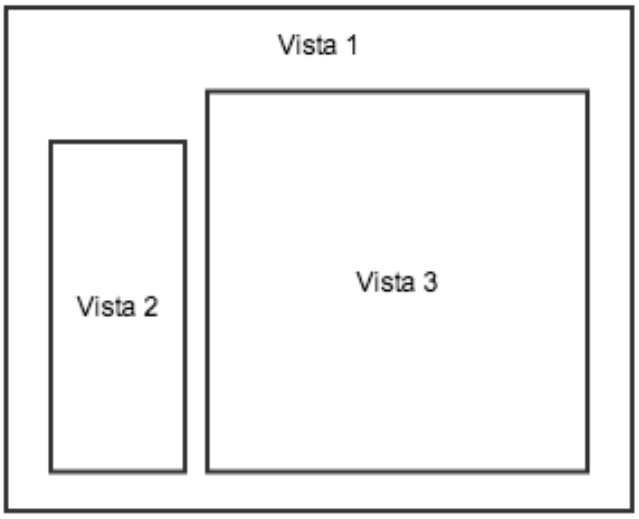

Figura 4.11: Diagrama de la jerarquía de las vistas

En esta figura 4.11 se puede ver cual es el resultado real de la jerarquía de vistas. Unas vistas superponen a otras creando de esta manera la interfaz. Como cabe imaginar, por ejemplo una acción en la *vista 3* puede requerir el cambio de dicha vista por otra.

Por lo tanto, la *vista 3* necesitará avisar a la vista de un nivel superior que la contiene, en este caso la *vista 1*, para que este realice la acción necesaria. Esto se traduce en que el controlador de la *vista 3*, envía un evento al controlador de la *vista 1*. El controlador de la *vista 1* elimina la *vista 3* y despliega otra en su lugar, junto a su controlador. Este último será ahora el encargado de enviar eventos al controlador de la *vista 1*.
Con este diseño jerárquico de vistas y controladores es como se construye la interfaz de la aplicación y su lógica. Obviamente la realidad es más compleja que simples controladores que gestionan vistas y se comunican entre ellos, pero sí es el núcleo del funcionamiento de la aplicación.
A continuación se va a mostrar una representación de las principales vistas de la aplicación, figura 4.12, de la que implícitamente se entiende que cada una lleva un controlador asociado, al igual que cualquier vista de la aplicación.

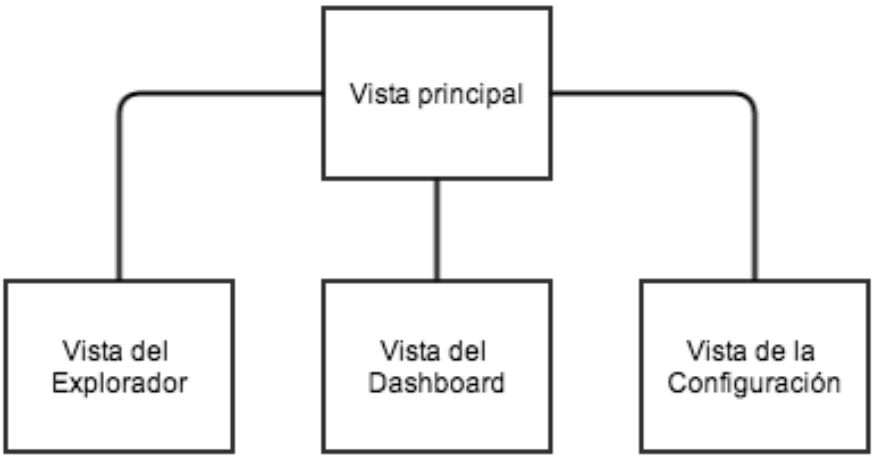

Figura 4.12: Diagrama de las principales vistas de la aplicación

La aplicación cuenta con la vista principal y a su vez 3 subvistas principales. Cada una de ellas representa a cada una de las principales funcionalidades de la aplicación que se han mencionado en la subsección 4.3.1. Por el mismo orden que en la figura 4.12, las funcionalidades son el explorador de los recursos del sistema a monitorizar, el dashboard para mostrar las métricas que se desean monitorizar y el configurador de la aplicación.
Las vistas representadas, a su vez contienen otras vistas con sus respectivos controladores. Se muestran a continuación cada una de ellas sin entrar en detalles de la composición de las vistas. Esto se explicará en la sección 4.4.

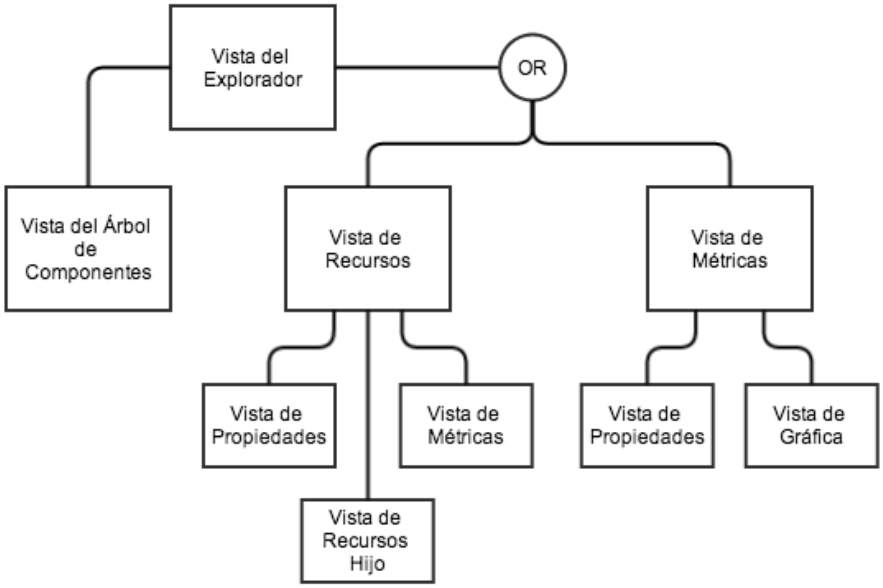

Figura 4.13: Diagrama de las vistas del explorador de recursos

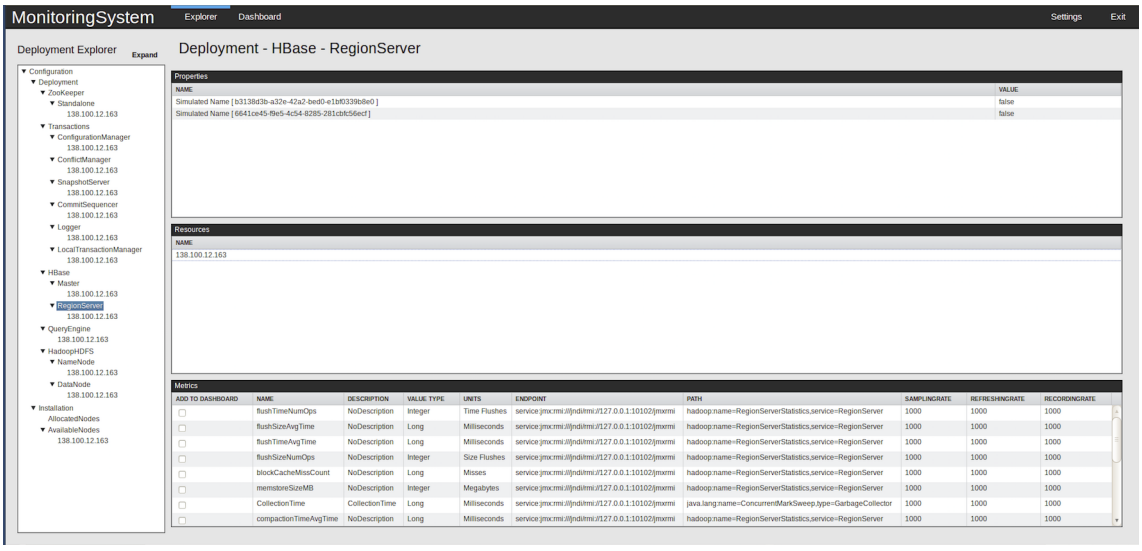

Figura 4.14: Representación real del diagrama de la fig 4.13

En las dos figuras 4.13 y 4.14, se puede comprobar el diseño de la aplicación y su representación real para el explorador de recursos. En ésta última representa a la vista de recursos, junto a sus tres subvistas y el árbol de exploración de recursos.

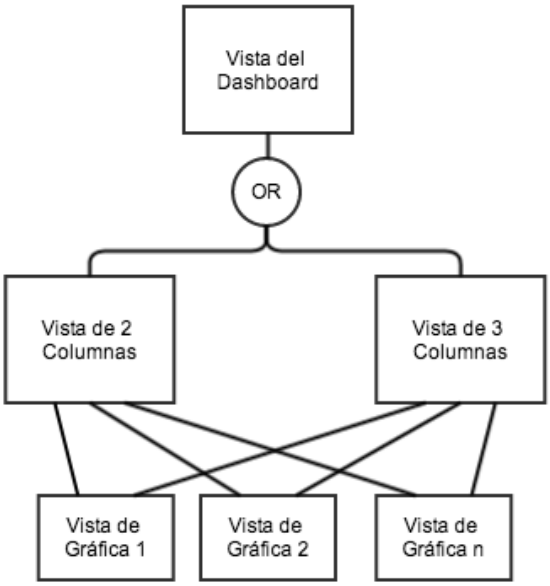

Figura 4.15: Diagrama de las vistas del dashboard

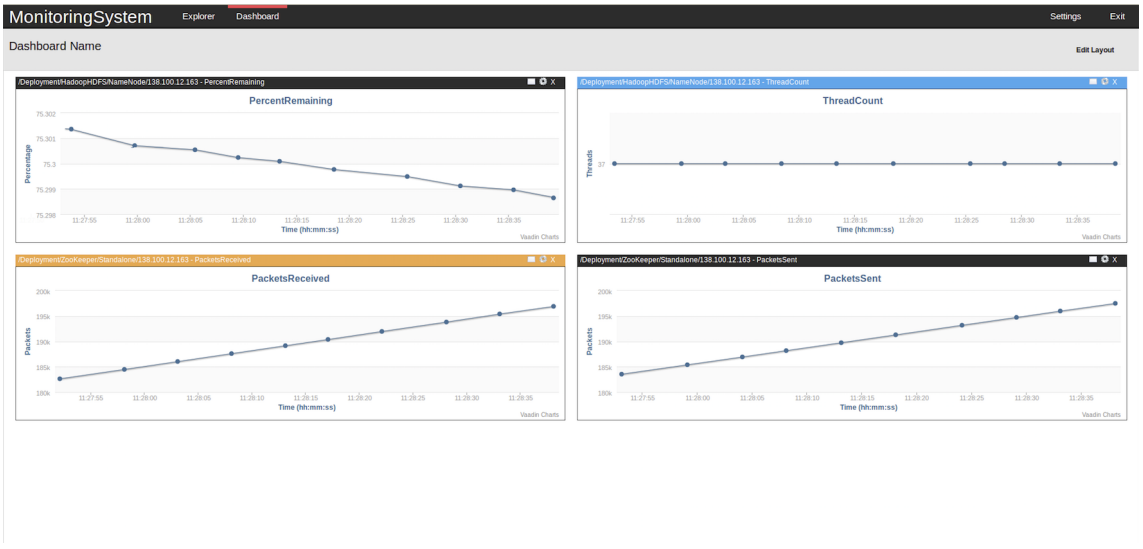

Figura 4.16: Representación real del diagrama de la fig 4.15

En las dos figuras 4.15 y 4.16, se puede comprobar el diseño de la aplicación y su representación real para el dashboard. La figura 4.16 representa a la vista de 2 columnas, junto a 4 gráficas.

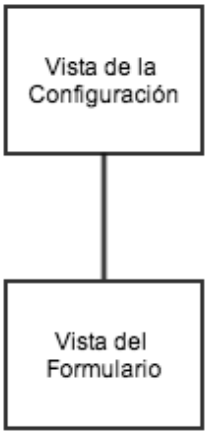

Figura 4.17: Diagrama de la vista del configurador

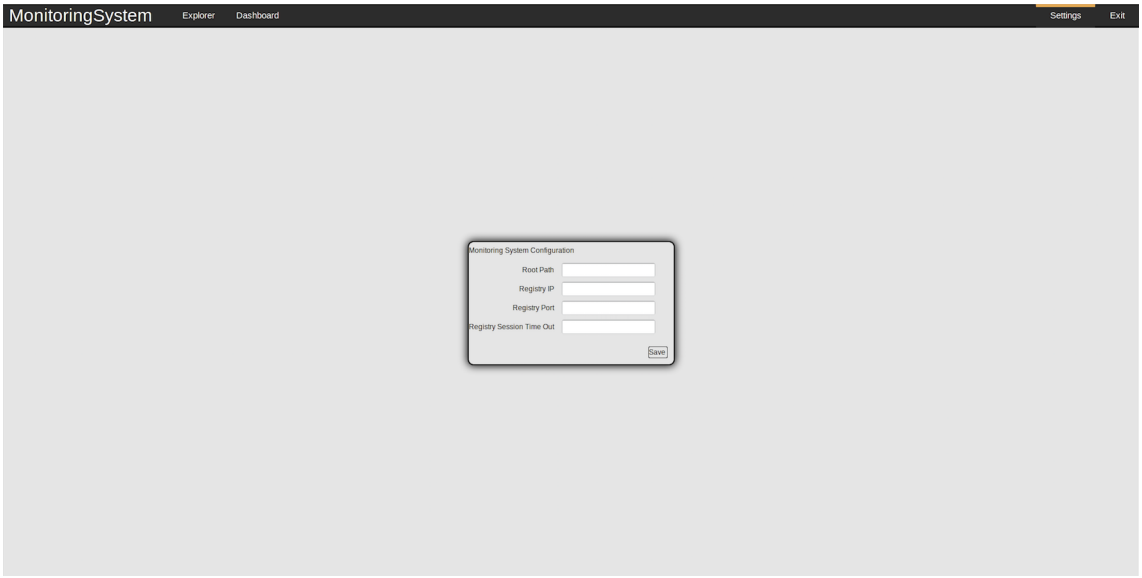

Figura 4.18: Representación real del diagrama de la fig 4.17

En las dos figuras 4.17 y 4.18, se puede comprobar el diseño de la aplicación y su representación real para el configurador de la aplicación.

Una vez se han visto las figuras anteriores, se tiene una idea de la estructura de la aplicación. Recordar nuevamente que aunque solo se muestran las vistas, cada una de ellas lleva siempre asociado un controlador.

4.3.3. Eventos

Para el apartado de eventos se ha empleado el patrón observador. Este patrón de diseño permite que un objeto cuente con una lista de objetos dependientes de él, a los que habitualmente se les conoce como *listeners* u *observadores*. En el supuesto de que el objeto principal necesite notificar cualquier tipo de evento a sus observadores, lo hará con la invocación de un método de la interfaz, la cual implementan los propios observadores. De esta manera se permite que un objeto pueda notificar a otro, de manera asíncrona, eventos que son de interés para este o estes últimos.
La aplicación cuenta principalmente con lo que podríamos calificar como tres tipos de eventos según su función y lugar en la aplicación, a pesar de ello su funcionamiento es el mismo. Los primeros están relacionados con el modelo, los segundos son eventos capturados por los elementos que componen las vistas de la interfaz, y manejados por los controladores de estas, los terceros son eventos entre los propios controladores de la aplicación.

Eventos relacionados con el modelo Como se ha visto con respecto al modelo de la aplicación, ésta cuenta con dos elementos que componen dicho modelo, la base de datos y el registro. Mientras el primero no cuenta con ningún tipo de eventos para notificar a la aplicación, el segundo sí lo hace. Inicialmente el registro se utiliza para que la aplicación de monitorización pueda obtener una imagen del sistema a monitorizar. Esta imagen permite que la aplicación muestre la jerarquía del sistema, además de proveer información para el acceso a la base de datos. Mencionar que el sistema a monitorizar es dinámico, lo que se traduce en que este puede sufrir cambios como la no disponibilidad de alguno de sus componentes, adicción dinámica de otros, etc. Para dar soporte a todo ello, la aplicación de monitorización podría encargarse cada cierto tiempo, de consultar el registro en busca de diferencias con respecto al modelo actual que la aplicación tuviera en ese momento. Como se puede deducir, esto sería una tarea totalmente ineficiente, dado que si hubiera un bajo nivel de actualizaciones, se emplearía capacidad de cómputo innecesariamente. La solución es que el modelo ejerza de notificador, y los controladores de listeners u observadores. Este se traduce en que los controladores se van a subscribir a los eventos que pueda generar el registro con respecto a las modificaciones del contenido de éste. En el momento en el que el registro genere un cambio, la aplicación será notificada mediante la invocación de un método que implemente la interfaz definida. Será el registro, el que se encargue de invocar este método, por cada uno de los listeners que tenga suscritos.
En la figura 4.19, se muestra quién es el controlador encargado de la recepción de dicho evento en la aplicación. Se ilustra con un breve diagrama de los controladores principales de la aplicación. Mencionar nuevamente que por cada vista de la aplicación existe un controlador, por lo que el diagrama es similar a los realizados para el subapartado anterior, en el que se mostraba la jerarquía de vistas. En la figura anterior se ve como el controlador encargado de la vista del árbol de componentes, es el listener u observador del registro. Este controlador, cuando inicia el conector para su conexión con el registro, se encarga de subscribirse a los eventos que el registro genera, con respecto a la modificación de la información del registro.

Después de recopilar la información del registro al inicio del sistema, además de gestionar los eventos que se producen en la vista del árbol, también está a la espera de eventos que alteren el modelo actual del árbol. Cuando recibe un evento de este tipo, o bien se encarga de eliminar un nodo del árbol junto a sus hijos, si los tiene, o de añadir uno nuevo, y en caso de que tenga hijos, solicitarlos.

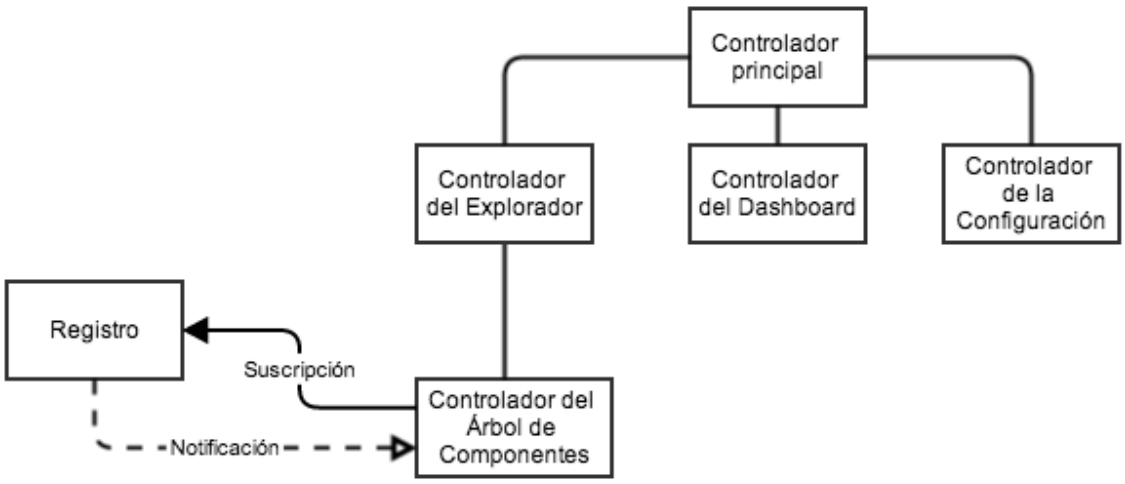

Figura 4.19: Diagrama de los controladores principales y patrón de eventos con el modelo

Eventos relacionados con las vistas Los eventos de la interfaz son generados por la interacción del usuario. Como se ha explicado, los eventos son generados a nivel de implementación por un objeto, y recibidos por otro u otros que están suscritos al primero. En este caso, los componentes que forman las vistas como: botones, etiquetas, paneles, etc; son lo que generan dichos eventos. Los controladores que manejan esa vista, la cual contiene a todos esos componentes, son los suscriptores de dichos eventos. Como se puede ver en la figura 4.20, un controlador responsable de una vista es listener de cada uno de los componentes que la forman, y tan solo de los que tienen capacidad para generar eventos; a la vez, que estos eventos sean de interés para la aplicación. Con esta explicación se empieza a vislumbrar que realmente una vista es un conjunto de componentes, ordenados de tal manera que formen la vista final. Dado que como se ve, realmente la interacción es entre los componentes de la vista y el controlador, por lo que la vista es como el molde que es rellenado de una manera ordenada, por componentes o vistas más pequeñas. A esta vista final se la conoce también como *composite*. Se hará un estudio más profundo de la formación de vistas en la sección 4.4.

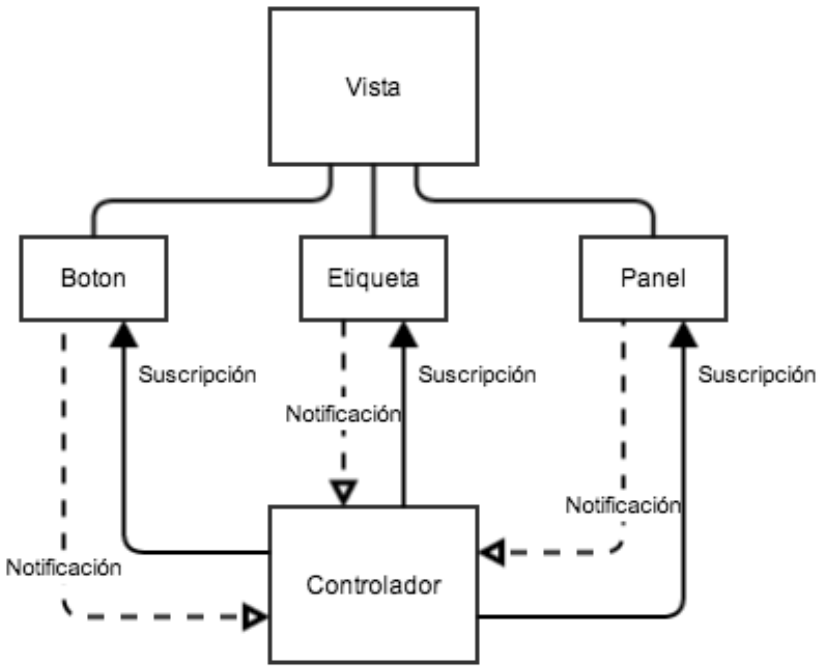

Figura 4.20: Diagrama del patrón de eventos entre la vista y el controlador

Eventos relacionados con los controladores El tercer tipo de eventos son los que se generan entre los propios controladores de la aplicación. En este caso, se han implementado tanto los emisores como los receptores. Estos son controladores que generan eventos y controladores que los reciben. Como ya se ha explicado anteriormente, la aplicación cuenta con un diseño jerárquico, lo que hace que unas vistas estén contenidas dentro de otras, de igual modo, los controladores cuentan a su vez con controladores de subvistas por debajo de ellos. Esto conlleva a que en ocasiones, los controladores tengan que compartir información entre ellos. Por ejemplo, en el caso de que dentro de una vista se pulse un botón, el evento de este botón es capturado por el controlador de dicha vista. Pero si dicha acción requiere que se cambie esa vista por otra del mismo nivel, esto conlleva que el controlador de la vista superior reciba el evento, y haga los cambios necesarios en la vista de su nivel para que esta contenga ahora la nueva vista.

En la figura 4.21 se puede ver el ejemplo descrito anteriormente. La vista de menor nivel contiene un botón, el cual genera un evento, este evento es capturado por su controlador, que a su vez genera un evento para ser comunicado al control de un nivel superior. Una vez que el controlador del nivel superior recibe el evento, éste se encarga de gestionar la vista para su actualización. Este tipo de eventos que se han descrito, son comunicaciones entre controladores para el envío de información. En el ejemplo anterior la notificación era simplemente que se había pulsado un botón. Como se puede imaginar, la información que puede contener el evento, además de la propia interacción con la interfaz, puede ser

de varios tipos dependiendo de la situación. Entre esta información se puede encontrar la información de una métrica concreta, path de un componente del árbol de recursos, etc. Otra cosa a tener en cuenta es la repercusión de un evento, es decir, los niveles de la jerarquía por los que se propaga. Se debe de tener en cuenta que una acción en la interfaz no implica solamente que el controlador de su nivel, o de un nivel superior, gestionan dicho evento. En muchos, estos controladores simplemente llevan a cabo la tarea de retransmitir este evento hacia un nivel superior, hasta que alcance un controlador que esté interesado por dicha información.

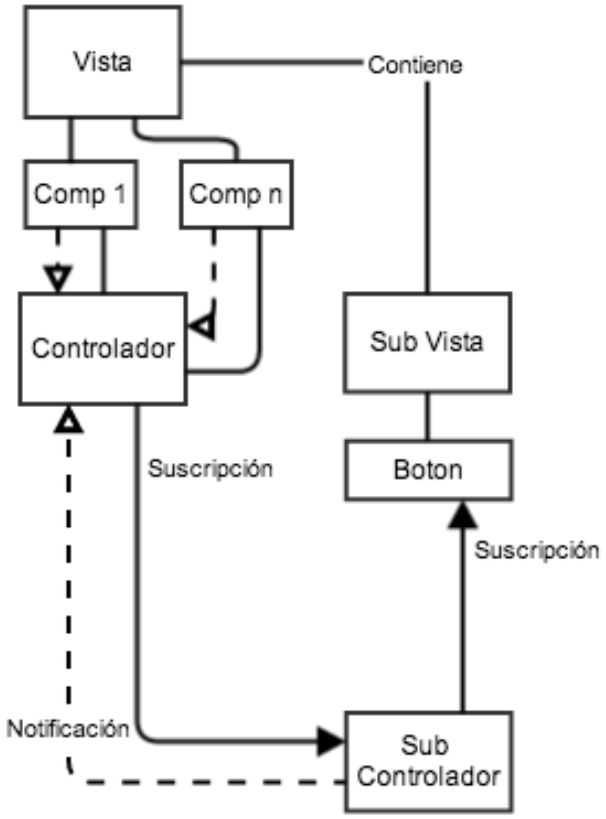

Figura 4.21: Diagrama del patrón de eventos entre los controladores

Un ejemplo de la repercusión y los niveles que puede recorrer un evento, es el caso de añadir una métrica de un componente del sistema para ser monitorizada en el dashboard. Para que la métrica pueda ser añadida en el dashboard, es necesaria información sobre dicha métrica. Esta información es proporcionada por el evento que se genera cuando se selecciona dicha métrica en el panel de métricas de un componente. Como se vió en las figuras 4.12,4.13 y 4.15, la vista de métricas estaba contenida en la vista de recursos, esta a su vez en el explorador, y a todo a su vez en la vista general. Esta vista general podía contener la vista del explorador o la del dashboard. Por lo tanto, para que el evento desde la vista de métricas, llegue a ser una notificación del dashboard, el número de controladores asociado a cada vista, debe ser por supuesto mayor que dos.

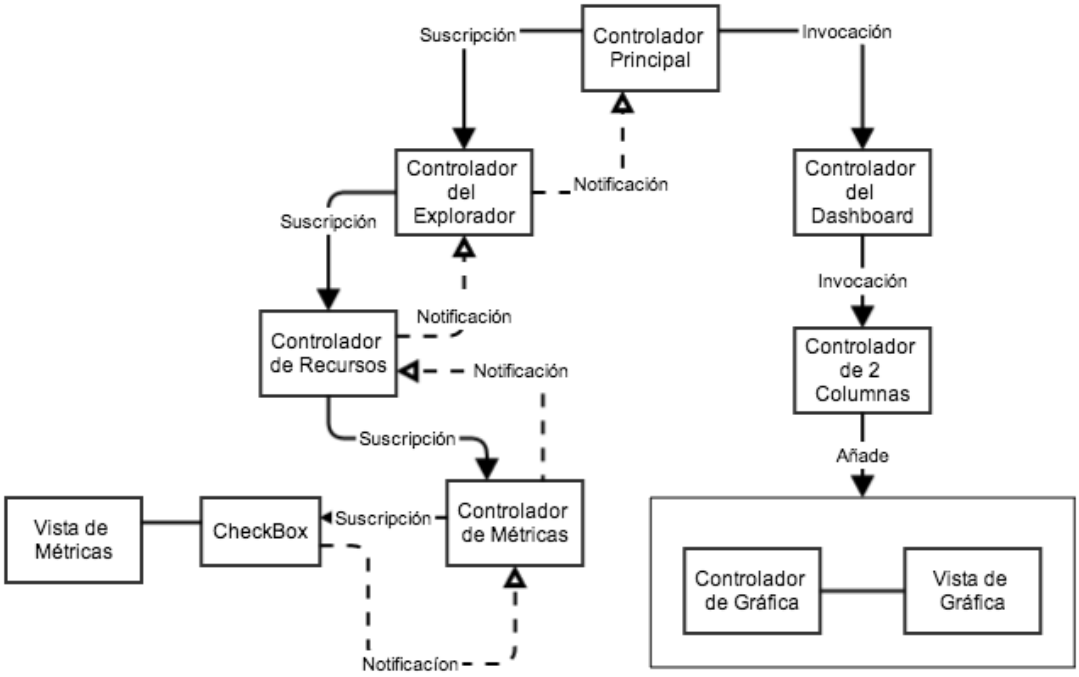

Figura 4.22: Diagrama de la propagación de un evento por la aplicación, desde el panel de métricas hasta el dashboard

En la figura 4.22, se puede ver la propagación del evento a lo largo de los controladores de la aplicación. El evento es generado cuando el usuario selecciona mediante un checkbox a una métrica de un componente del sistema a monitorizar. Este evento es propagado desde abajo hacia arriba de la jerarquía. Una vez que llega al controlador más alto de la jerarquía, mediante invocaciones a controladores de la jerarquía de arriba a abajo, se termina añadiendo en el dashboard la métrica en forma gráfica.

4.3.4. Flujo de funcionamiento

En esta subsección se van a dar dos visiones del funcionamiento de la aplicación. La primera será una visión con respecto a las clases, representadas por las vistas y sus controladores. De estas se dirá cuales se instancian al desplegar la aplicación, y en qué orden, y cuales lo hacen más tarde, a medida que se interacciona en la aplicación. Además, cuales son mantenidas en memoria y reutilizadas mediante la modificación de sus datos, y cuales se generan cada vez que hay datos nuevos. La segunda dará una visión de cómo fluye la información por la aplicación, desde el inicio del sistema hasta que este está haciendo uso de todas sus funcionalidades. Se explicará qué información necesita la aplicación para iniciarse con un despliegue concreto, que información recaba sobre este despliegue, y cómo a partir de ahí se solicita información para su monitorización.

Flujo de la aplicación El funcionamiento de la aplicación se basa en la creación de vistas y controladores, para formar la aplicación. Decir que estas vistas y estos controladores son instancias de clases definidas en el desarrollo. La explicación se va a realizar representando a estos objetos o instancias de las clases, como vistas y controladores. Como se ha dicho en numerosas ocasiones, cada vista siempre tiene asociado un controlador y viceversa. Por lo tanto, y debido a su naturaleza jerárquica, la aplicación cuenta con una vista y un controlador principal, a partir de los cuales cuelga toda la aplicación. A partir de ahí, por cada nivel de la aplicación se va realizando el proceso que se detalla a continuación.
Cuando se inicia una vista, se inicia a su vez su controlador. A este último se le pasa una referencia de la vista, para que así se pueda subscribir a los eventos que los componentes de la vista generan. A partir de ahí, el controlador se encarga de ir creando y añadiendo las subvistas necesarias, para continuar componiendo la jerarquía. Por ejemplo para el controlador del explorador, crear las vistas de las propiedades, los recursos hijo y las métricas. De qué manera se añaden unas vistas a otras, se verá en la sección 4.4. El controlador también se encarga de ir creando los correspondientes controladores por cada una de ellas, y suscribiéndose a sus eventos en caso de que esto sea necesario. Por ejemplo, en el caso de que una subvista tenga que notificar algo a la vista de nivel superior, se hace a través de eventos entre los controladores, como se ha explicado en la subsección 4.3.3. Este proceso se realiza de manera recursiva a lo largo de todos los niveles de la aplicación. Este proceso, al que podríamos denominar como incial, se hace por cada uno de los componentes que son creados tan solo una vez en la aplicación, y son reutilizados para mostrar la distinta información. Estos se crean y se almacenan en memoria, a la espera de que se solicite que se muestren por pantalla. Para los componentes que se crean dinámicamente, el proceso de creación es el mismo, tan solo que no se crean al inicio, sino cuando se requiere por el usuario.

En la figura 4.23 se pueden ver todos los componentes de la aplicación de monitorización, entendiéndose que por cada componente se representa una instancia de su vista y otra instancia de su controlador. Decir que los componentes de color blanco, son objetos que se inician tan solo una vez, y son mantenidos en memoria por la aplicación durante una sesión de monitorización. En cambio, los componentes de color azul representan a instancias u objetos que se crean dinámicamente en el transcurso de la aplicación. Los componentes de color blanco se crean al iniciar la aplicación, y se destruyen cuando el recolector de basura de Java los elimina, una vez se ha cerrado la sesión de la aplicación. Por el contrario, los componentes de color azul se crean cuando el usuario los solicita, y se eliminan cuando el usuario lo requiere. Por ejemplo la gráfica de las métricas se crea al solicitar la visión de una métrica, y se elimina nada más cambiar de vista. El caso de las gráficas del dashboard es un poco más complejo. Estas se crean o se eliminan cuando el usuario lo solicita, o permanecen en memoria hasta que se cierra la sesión y terminan por ser eliminadas por el recolector de basura.
Los componentes de color blanco son mantenidos en memoria, y según los requiera o no el usuario, se muestran en la interfaz. Por ejemplo, en caso de que nos encontremos visualizando en el dashboard una serie de métricas, toda la información del explorador junto al árbol de recursos es mantenida en memoria. En el momento que el usuario decide

dejar de visualizar el dashboard para pasar al explorador, el propio dashboard pasa a un segundo plano, y el explorador a un primer plano, traído directamente de la memoria. A nivel de desarrollo, esto se realiza agregando y desagregando unas vistas de otras, como se verá en la sección 4.4.

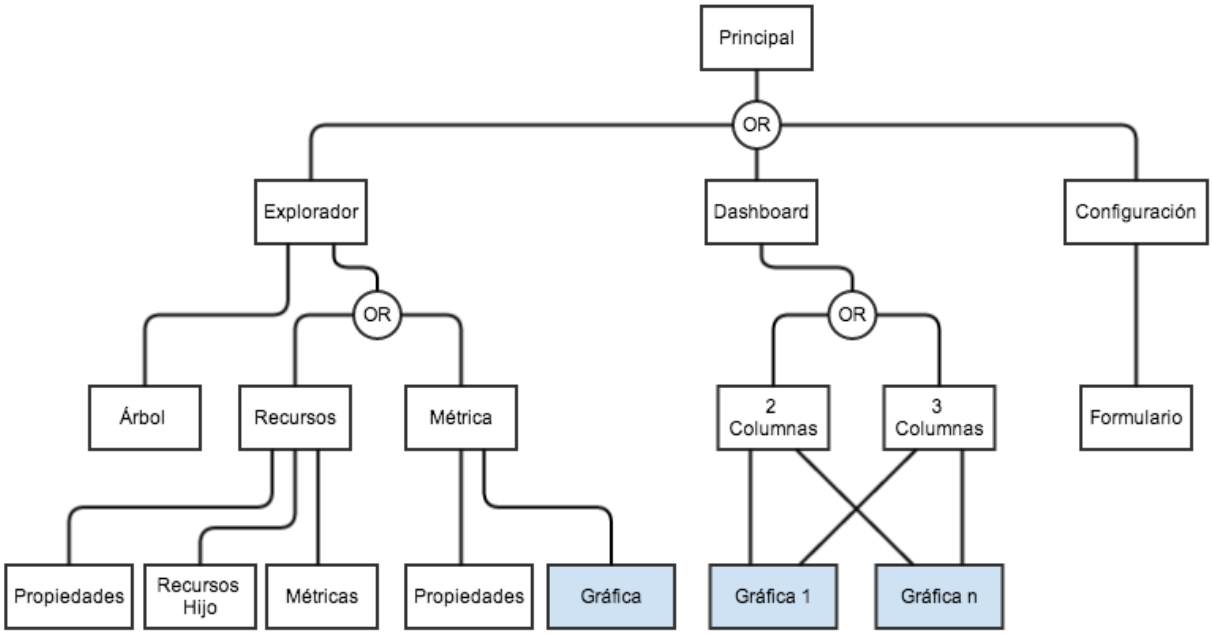

Figura 4.23: Diagrama de los componentes según su reutilización

Flujo de la información La aplicación comienza con la creación de las vistas y los controladores que perduran a lo largo de una misma sesión de la aplicación, como se ha explicado anteriormente. Una vez que la aplicación se muestra por pantalla, lo primero que se solicita al usuario es información. En este punto es en donde comienza el flujo de información de la aplicación. La información solicitada, son datos a partir de los cuales la aplicación puede conectarse con el modelo. Estos datos son:

- Dirección IP del registro.
- Puerto del registro.
- Directorio del registro donde se encuentra la información del sistema a monitorizar.
- Time out del registro.

Una vez que la aplicación cuenta con esa información, ésta crea un conector por cada uno de los dos componentes del modelo. A continuación, la aplicación comienza a recorrer el árbol de directorios del registro, en el cual se encuentra la configuración del despliegue del sistema a monitorizar. El objetivo es realizar una copia de dicho árbol, para

poder ser mostrada al usuario, sin tener que estar en cada momento solicitándolo al registro. Esta copia se almacena en una estructura de datos adecuada para el almacenamiento de estructuras en forma de árbol. Como se ha explicado en el capítulo 3, en Vaadin existen una serie de clases que permiten almacenar distintos tipos de estructuras de datos. Dado que muchas vistas (tablas, árboles, gráficas, etc) necesitan de información que mostrar, estas clases están diseñadas para dar soporte al almacenamiento de dicha información, con un formato adecuado para ser visualizada. En este modelo de árbol, la información se almacena en una estructura que es soportada por el componente visual del árbol, en el cual se muestra la jerarquía de los componentes del sistema a monitorizar. Como ya se explicó en la subsección de eventos 4.3.3, el registro, en caso de que se modifique, notificará mediante un evento a la aplicación para que ésta tenga consistencia en su árbol. Una vez en su poder la información de cada uno de los nodos del árbol del registro, junto con cierta metainformación disponible a nivel local en la aplicación, ésta muestra el explorador de recursos con el árbol del sistema a monitorizar. Esta metainformación permitiría la identificación unívoca de cada uno de los componentes del sistema a monitorizar, para poder ser obtenidos a través del conector de la base de datos de manera correcta.

Como se describió en la subsección de consistencia del sistema 4.2.2 esta metainformación consiste en:

- Direcciones IP de los componentes físicos donde están ejecutándose los componentes lógicos. Permite discernir entre dos componentes lógicos iguales, por la máquina en la que se están ejecutando.

- Referencia mediante un nombre y una versión, a los ficheros de definición de métricas asociados a cada uno de los componentes lógicos y físicos, para especificar el tipo de métricas que se solicitan.

Esta información, junto a los paths del árbol de directorios, hacen que las invocaciones a la base de datos sobre determinados componentes físicos o lógicos, sean inequívocas. A partir de este momento, el usuario puede comenzar a recorrer el árbol de componentes y a seleccionar uno determinado para su visualización. Cuando el usuario selecciona un componente, todo viene orquestado por la navegación por el árbol. Como se ha explicado, este árbol contiene toda la información necesaria para el acceso a la base de datos. Por lo tanto, cuando se selecciona un componente, el propio controlador del árbol genera un evento a su controlador superior (explorador), para que éste gestione las vistas necesarias junto a sus controladores. El resultado final es el despliegue de la vista de recursos, figura 4.14. Esta vista contiene tres subvistas: propiedades, recursos y métricas. La vista segunda se actualizará con la información procedente del árbol, ya que tan solo muestra los hijos de dicho componente. Para las propiedades y la métricas se accede a las base de datos. Con respecto a las métricas, para el acceso tan solo es necesaria la metainformación del nombre y la versión de los ficheros, debido a que todos los componentes del mismo tipo comparten las mismas métricas. Otra cosa es el valor de estas.

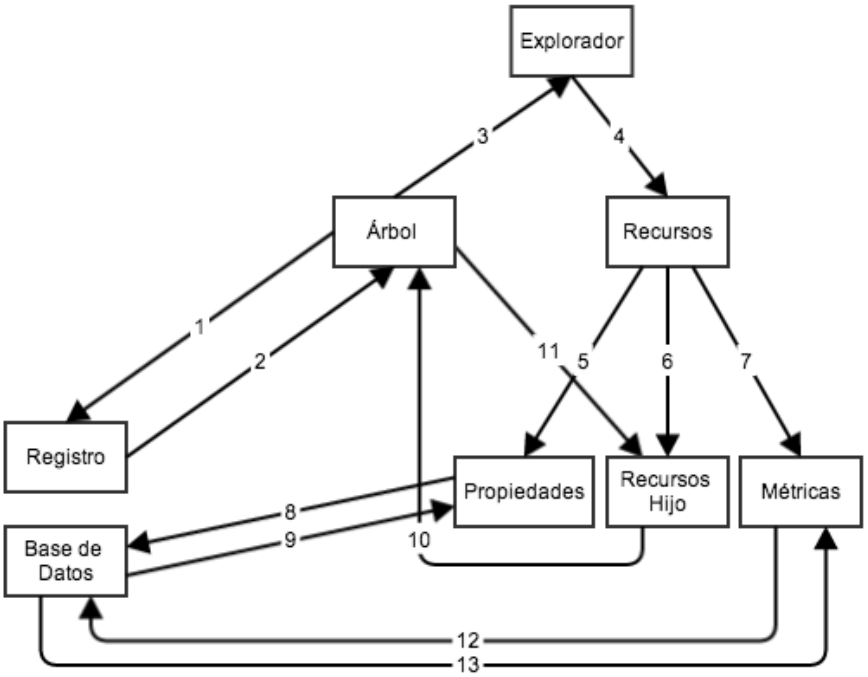

Figura 4.24: Orden del flujo de información para visualizar un componente

A partir de la figura 4.24 se va a especificar qué información se maneja en cada paso, para la visualización de un componente del sistema a monitorizar:

1. El árbol solicita los path hijos, para un path padre, si los tiene.
2. El árbol recibe los hijos y los almacena junto a la metainformación (IP, nombre y versión). Realiza este paso junto al paso 1 hasta que se recorre todo el árbol del registro.
3. En el árbol se selecciona un componente del cual su información se envía al explorador. En esta información se encuentra el path, la IP, el nombre y la versión.
4. El explorador envía la información al recurso.
5. El recurso envía la IP y el path a las propiedades.
6. El recurso envía el path a recursos hijos.
7. El recurso envía el nombre y la versión a las métricas.
8. Las propiedades solicitan a la base de datos la información a partir del path y la IP, las cuales hacen al componente único.
9. La base de datos le devuelve a las propiedades la información para que esta se muestre.

10. El recurso hijo le solicita al árbol que le devuelva el nombre de sus hijos para un path dado.

11. El árbol le devuelve los paths absolutos de sus hijos, si los tiene.

12. Las métricas solicitan a las base de datos la información a partir del nombre y la versión que identifica al conjunto de métricas de un componente.

13. La base de datos devuelve una lista con dichas métricas.

La otra posible solicitud de información se produce con la visualización de la vista de una métrica, o la adicción de esta al dashboard. Desde el panel de recursos visto antes, se puede seleccionar bien la visualización de una métrica o que ésta sea añadida al dashboard. En ambos caso, se debe de identificar a dicha métrica, únicamente para poder obtener su información del componente correcto. Dado que en ambas posibilidades ya se cuenta con la identificación de la métrica, lo que hace falta es identificar al componente. La identificación del componente se realiza a partir del path y de la IP. Esta información es obtenida del árbol. Una vez con la información de la métrica concreta, junto a la identificación del componente, se puede comenzar a solicitar a la base de datos los valores de dicha métrica a lo largo del tiempo. Por lo tanto, las gráficas que se muestran en la aplicación solicitan la información de la base de datos a través de un objeto formado por la IP y el path, junto a un objeto que representa a una métrica.

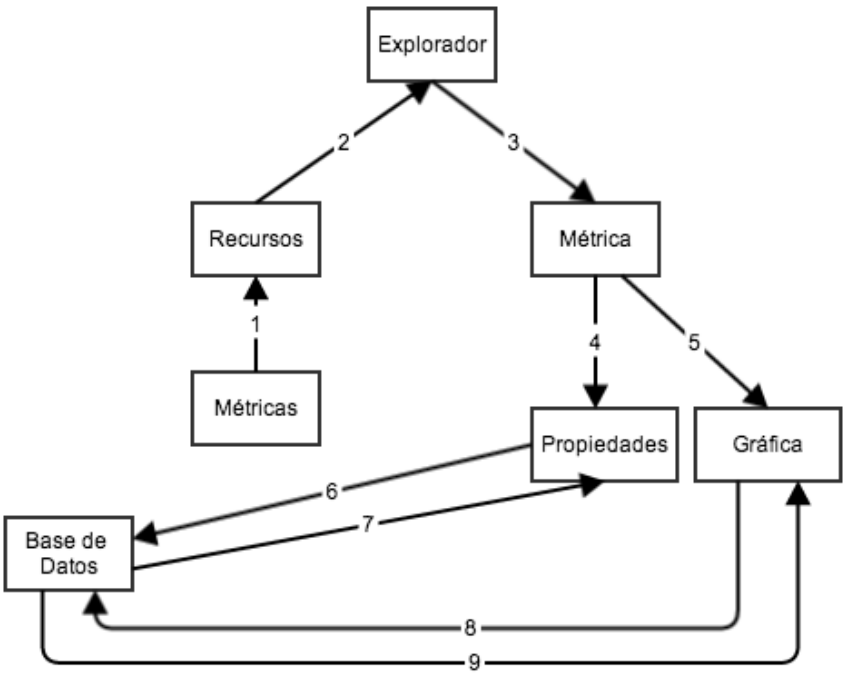

Figura 4.25: Orden del flujo de información para visualizar una métrica

A partir de la figura 4.25 se va a especificar qué información se maneja en cada paso, para la visualización de una métrica:

1. Una métrica es seleccionada, y ésta envía la información de que métrica es, al componente recursos.
2. El componente recursos envía la información al explorador de la métrica, junto a la IP y el path que identifican al componente del sistema a monitorizar.
3. El explorador envía la misma información que recibe al componente métrica.
4. El componente métrica le envía el nombre y la versión a las propiedades.
5. El componente métrica le envía el nombre, la versión, la IP y el path a la gráfica.
6. El componente propiedades le solicita a la base de datos la información de las propiedades de la métrica.
7. La base de datos le responde con una lista de dichas propiedades de la métrica solicitada.
8. La gráfica le solicita a la base de datos el valor de la métrica a partir del nombre, la versión, la IP y el path.
9. La base de datos le devuelve a la gráfica el valor de la métrica más reciente que tenga. Los procesos de los puntos 8 y 9 se repiten tantas veces como tiempo se desee visualizar la métrica.

4.4. Vistas de la aplicación

En esta sección del trabajo se va a explicar todo lo relacionado con la parte visual de la aplicación. Esta comprende la creación de las vistas, la customización de estas y como son añadidas a la interfaz. Los componentes que forman estas vistas o composites van desde paneles, pasando por botones y textfiles, hasta gráficas. Por lo que para comenzar se hará una introducción a la formación de vistas y a su funcionamiento. A continuación se desarrollará un caso concreto de las vistas: las gráficas, debido principalmente a su importancia en toda la aplicación, dado que es el vehículo esencial para la monitorización. Finalmente se describirá el uso de estilos para la customización de la interfaz de la aplicación.

4.4.1. Composites

Como se ha dicho en la sección 3.2, las vistas en Vaadin están compuestas por dos tipos de componentes. El primero de ellos abarca a todos los componentes finales con los que el usuario puede interaccionar: botones, tablas, etiquetas, etc. El segundo abarca a todo tipo de layouts o paneles en los que se pueden posicionar cada uno de los componentes finales. Como cabe imaginar, un componente final no puede contener dentro de él ni a un panel, ni a otro componte final. Todo lo contrario ocurre con los paneles, los cuales pueden contener tanto a otros paneles, como por supuesto componentes finales. Así es como se forma la jerarquía de las aplicaciones en Vaadin, paneles que contienen a otros paneles y componentes finales. A cada uno de estos niveles de la jerarquía, representados por una vista, se les llama también composites. Desde una perspectiva más cercana al código, un composite no es más que una clase que representa a la vista de un nivel. Una aproximación a esta visión se proporcionó en la subsección 4.3.2. Aquí se proporcionaba una visión de la aplicación jerárquica, la cual permitía intuír la formación de las vistas en la aplicación, y la magnitud real de estas. De todas formas, la visión que se proporcionaba, no dejaba ver la dimensión real de la aplicación, dado que no se diferenciaban entre los dos tipos de componentes de las vistas, ni en que lugares estaban estas situadas.
En la figura 4.26, se puede ver que la dimensión de las vistas de la aplicación, es algo más grande de lo visto anteriormente. Remarcar nuevamente que las vistas están formadas por dos tipos de componentes, pueden ser: paneles y componentes finales. En la figura se puede ver esta diferenciación, por la variación entre los colores blanco y azul claro. El blanco representa a los paneles contenedores de otros paneles y componentes finales, mientras que el azul claro representa a dichos componentes finales. Mencionar que estos últimos se encuentran distribuídos a lo largo de toda la jerarquía, es decir, puede haber componentes finales tanto en el primer nivel como en el último. Esto se ve perfectamente reflejado en la figura 4.27. La única posibilidad inexistente, es que haya un layout en un nivel inferior a un componente final, en la misma rama de la jerarquía. Es decir, que un componente final contenga a un layout.

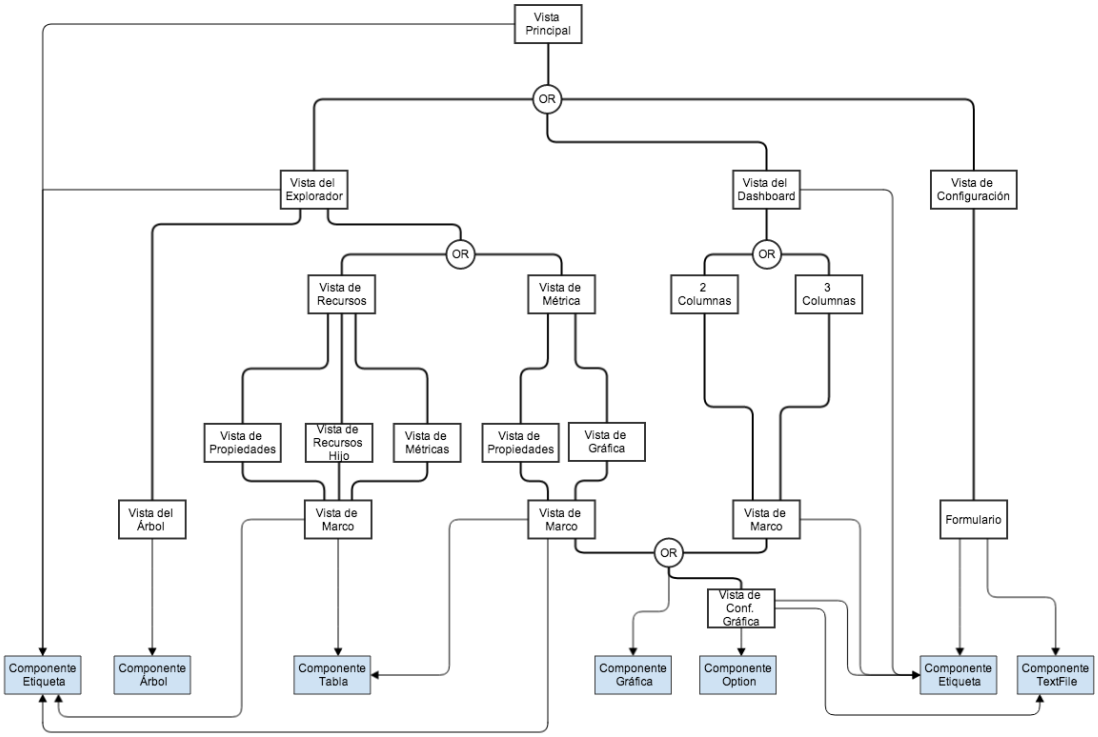

Figura 4.26: Diagrama de las vistas de la aplicación

Para la composición de la aplicación se han empleado tan solo un conjunto de paneles y de componentes finales de todos los que proporciona Vaadin. Estos se describen a continuación, junto a la función que desempeñan.

Primero se enumeran los tipos de paneles o layouts. A pesar de que Vaadin cuenta con unos diez tipos de paneles como se vió en el capítulo 3, tan sólo se han empleado los cuatro más generales.

- **Layout Vertical**. Permite posicionar otros layouts o componentes finales en orden vertical. Habitualmente tiene tamaño fijo a lo ancho e indefinido a lo largo.
- **Layout horizontal**. Permite posicionar otros layouts o componentes finales en orden horizontal. Habitualmente tiene tamaño fijo a lo largo e indefinido a lo ancho.
- **Layout Absoluto**. Permite posicionar otros layouts o componentes finales en una posición arbitraria dentro del panel. Siempre tiene tamaño fijo tanto a lo largo como a lo ancho.
- **Panel**. Permite posicionar otros layouts o componentes finales tanto en orden vertical como horizontal. Además, al contrario que con los layout vertical y horizontal, en caso de que los componentes de su interior sean de mayor tamaño que él, despliega un scroll en horizontal, vertical o en ambos sentidos; para poder visualizar así todo el contenido de su interior.

Ahora se van a enumerar los componentes finales empleados. Vaadin cuenta con innumerables componentes. Además existe la posibilidad de crear nuevos a partir de cero o de modificar los ya existentes. En el desarrollo se ha empleado tan solo un subconjunto de todos los componentes nativos de Vaadin.

- **Etiqueta**. Se emplea para mostrar texto que no se puede editar.
- **TextFile**. Permite que el usuario introduzca información. También se puede mostrar información a través de él.
- **Option**. Permite en una lista de elementos seleccionar uno de ellos, o varios. Según como se configure.
- **Tabla**. Permite mostrar información organizada por filas y columnas.
- **Árbol**. Permite representar de una manera natural información jerárquica entre la que hay relaciones.
- **Gráfica**. Permite la visualización de información numérica en una o dos dimensiones. Este punto se tratará en mayor profunidad en la sección 4.4.2.

En la figura 4.26, se pueden ver reflejados el uso de cada uno de los componentes finales en la aplicación, y en que vistas o composites han sido usados. Los layouts o paneles no se reflejan en esta figura, debido a que la mayoría de las vistas no están tan solo compuestas por un solo panel. En el caso de la figura 4.27 se puede comprobar como están distribuídas las vistas a lo largo de la vista final del explorador. Se puede ver una imagen real del explorador en la figura 4.14. Si nos fijamos por ejemplo en el nivel de la jerarquía que ocupa la vista del explorador, se ve como hay tres etiquetas en la parte superior de la vista. Cada una de estas etiquetas está en una posición fija. Además a su vez contiene a las vistas del Árbol y a la de los recursos. Para ellos se necesitan layouts sobre los que se puedan posicionar estas vistas y estos componentes finales. En la figura 4.28, se pueden comprobar los layouts de los que está formada la vista del explorador. Esta vista junto al resto de las de la aplicación han sido creadas con el editor visual que Vaadin proporciona como plugin para eclipse. En la figura 4.29 se puede ver una imagen de este editor, junto a la edición de la vista del explorador.

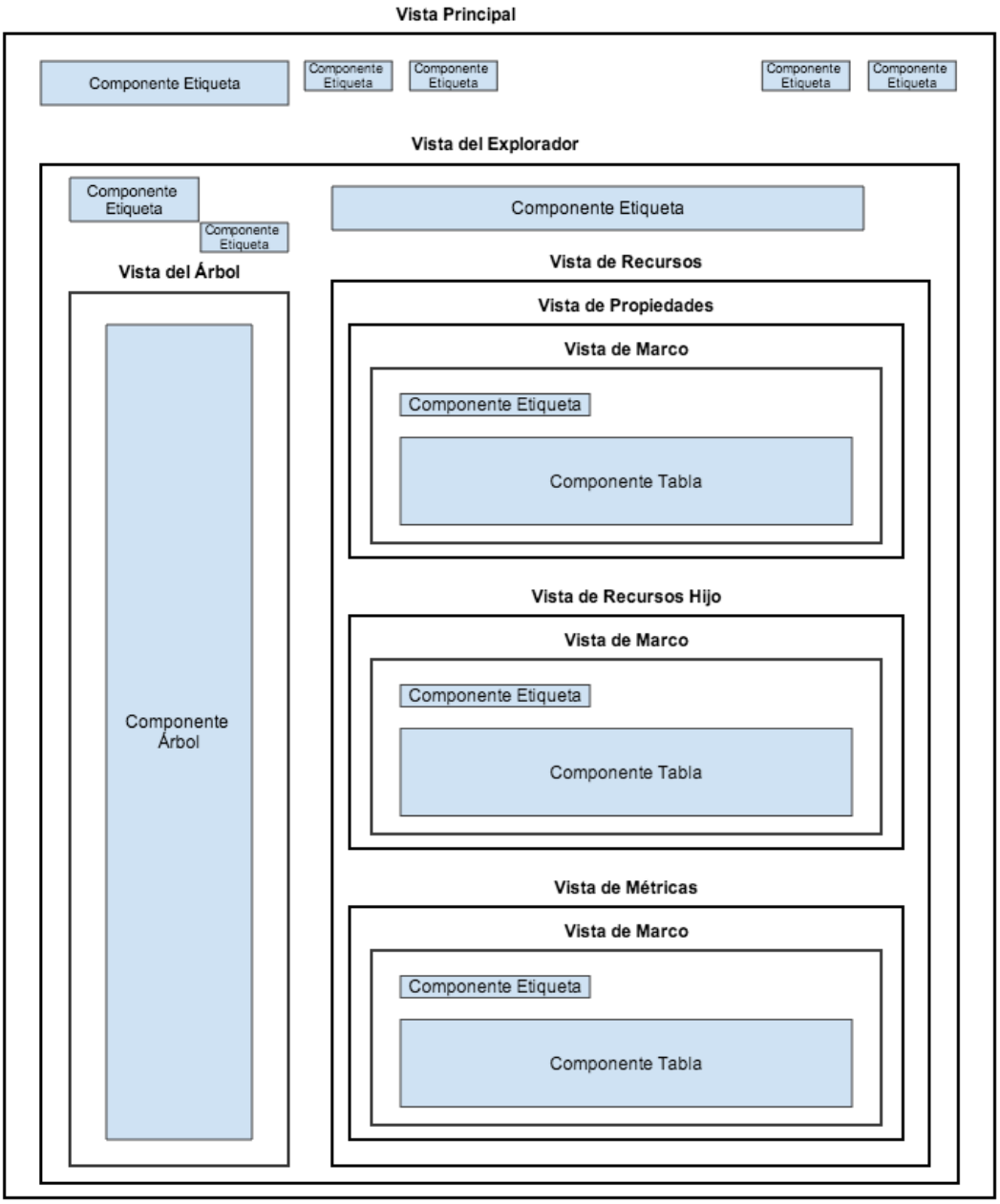

Figura 4.27: Diagrama de la vistas del explorador

Desde una perspectiva más cercana al funcionamiento de la aplicación, se va a tratar de explicar el funcionamiento de la misma a partir de la exposición anterior. Es decir, la vista de un nivel está compuesta por componentes finales y demás layouts, que permiten el posicionamiento de otras vistas. Estos layouts vacíos, son los puntos de conexión entre vistas, y también el punto de unión de la jerarquía. La jerarquía se forma a medida que se van desplegando vistas dentro de otras vistas, formadas por layouts y componentes finales. Decir que estas conexiones entre vistas son dinámicas. En la ejecución de la aplicación, un mismo layout puede contener dos vistas distintas.

Por ejemplo, en el caso del explorador, este siempre contiene al árbol, pero también a la vista de recursos o métricas. Por lo tanto, como se puede ver en la figura 4.28, el layout absoluto más grande que se ve en la imagen, será el que varíe y contenga a la vista de recursos o métrica, según el usuario lo solicite.

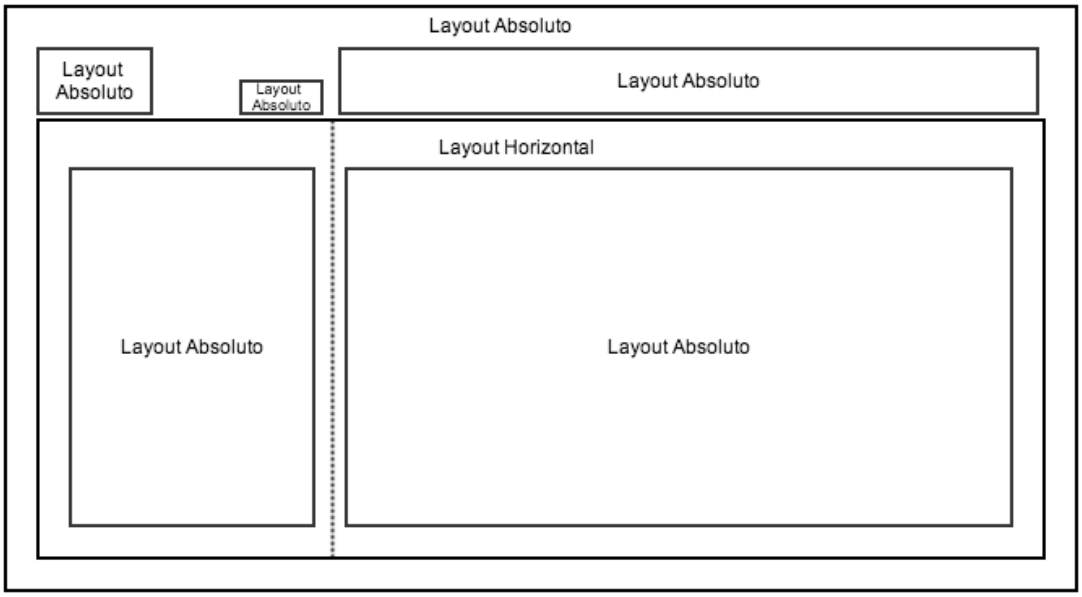

Figura 4.28: Layouts de la vista del explorador

Hasta ahora se ha visto el uso de los layouts absoluto y horizontal. Decir que los otros dos (vertical y panel) también son usados en otras vistas de la aplicación. Con especial representación de cada uno en dos vistas muy importantes. El panel se usa como layout principal de la vista del árbol, y como contenedor del componte final del árbol de recursos. Esto es debido principalmente a que el tamaño del árbol siempre es desconocido, tanto a lo largo como a lo ancho, según el tamaño de la jerarquía que represente. El panel vertical tiene un claro uso en la vista del dashboard. En este hay dos vistas disponibles (vista 2 y 3 columnas) en las cuales su función es la de ir añadiendo gráficas en orden descendente o vertical.

Se debe tener en cuenta que a lo largo de la aplicación se usan muchas vistas o composites iguales, pero en diferentes niveles y lugares de la jerarquía de las vistas. Principalmente debido a que la función que desempeñan es la que se necesita en estos diferentes lugares. El ejemplo más representativo de esto es la vista de marco. Esta se puede ver en la figura 4.26 como está presente en diferentes lugares de la jerarquía. Su principal función es la de acoger en su interior a un componte final y mostrar en su encabezado cierta información relativa a dicho componte. Existen dos versiones de este marco. Una que tan solo cuenta con información relativa a lo que contiene, y otra que además de esto cuenta con tres botones. El primero se emplea cuando el marco contiene al componente final tabla, y el segundo cuando contiene al componente final gráfica.

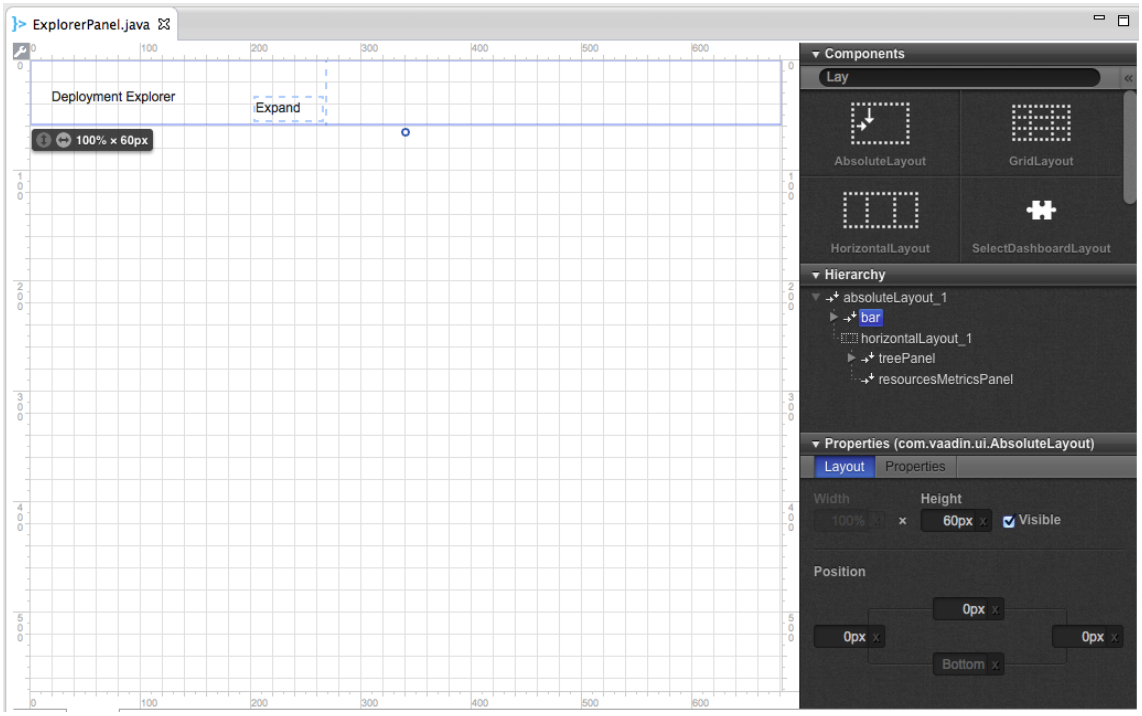

Figura 4.29: Herramienta de Vaadin para la edición visual de las vistas

En la aplicación no se han empleado los botones por defecto de Vaadin por motivos de estética. En su lugar se han empleado layouts con una etiqueta en su interior, lo cual es mucho más modificable visualmente. Con respecto a los marcos, se puede ver un ejemplo en la figura 4.30 en la que el marco pertenece a la vista del dashboard. Como se puede ver, en su interior contiene un componente final de gráfica, y en su encabezado, además de información se despliegan una serie de botones.

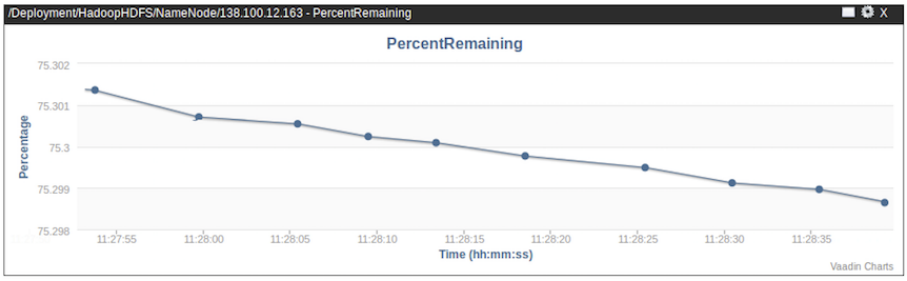

Figura 4.30: Vista de marco con gráfica en el interior

Ahora que ya se conoce perfectamente la estructura de las vistas y la creación de estas, se va a explicar brevemente la funcionalidad, y en qué lugares se emplean los componentes finales. Uno de los principales son las gráficas. De estas se hablará en la subsección 4.4.2. En la figura 4.26 ya se ha visto de color azul claro qué componentes finales se emplean y en qué vistas. De todas formas, ahora se va a mostrar un ejemplo de cada uno de ellos, y a explicar un poco las funcionalidades que estos proporcionan.

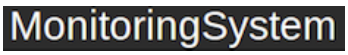

Figura 4.31: Ejemplo de etiqueta en la aplicación

Etiquetas Las etiquetas se emplean para mostrar texto que no puede ser modificado. En la aplicación se han formado los botones a partir de una etiqueta y un layout. Además, se han empleado como contenedores de texto para encabezados, como el nombre de la aplicación, texto del marco, etc. Las funcionalidades que se han empleado de esta etiqueta han sido la modificación de su contenido a mostrar. Se puede ver un caso de uso en la figura 4.31.

Figura 4.32: Ejemplo de TextFile en la aplicación

TextFile Los TextFiles se emplean para introducir texto que es necesario para el funcionamiento de la aplicación. En las vistas de la aplicación, se emplean en dos partes. La primera de ellas en la vista de configuración del sistema. A través de ellos se introducen los datos, para especificarle a la aplicación donde se encuentra el modelo. La segunda de ellas es en la vista de configuración de las gráficas. A traveś de los TextFiles se introduce información como: velocidad de muestreo, número de valores en la gráfica, etc. Las funcionalidades empleadas han sido las de la lectura del contenido escrito por el usuario. Se puede ver un caso de uso en la figura 4.32.

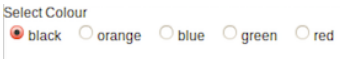

Figura 4.33: Ejemplo de Option en la aplicación

Option Los Option ofrecen una serie de etiquetas con unos marcadores, a través de las cuales se puede seleccionar una opción o varias. Dentro de la aplicación, se emplean en la vista de configuración de las gráficas, para seleccionar el color del marco que contiene a esa gráfica. Las funcionalidades empleadas han sido la lectura de la información de qué opción se encontraba seleccionada. Se puede ver un caso de uso en la figura 4.33.

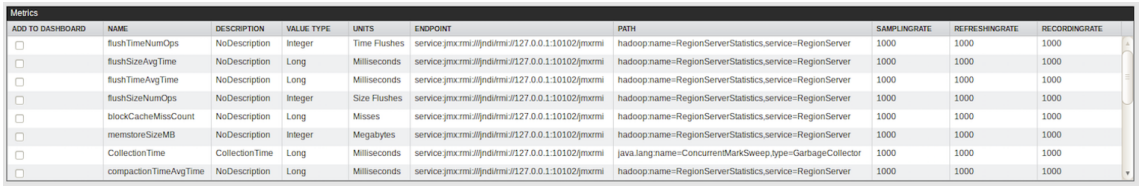

Figura 4.34: Ejemplo de tabla en la aplicación

Tabla Las tablas permiten visualizar información organizada por filas y columnas. Estas se han empleado en las vistas de propiedades, recursos hijos y métricas. De las funcionalidades que proporcionan se han empleado varias. La primera y más obvia la muestra de información. La segunda ha sido la capacidad para seleccionar un elemento de la tabla, mediante el cual se genera un evento. Finalmente la tercera, la capacidad de ordenar las columnas por orden alfabético. Decir que las tablas cuentan con un objeto asociado a modo de contenedor, el cual permite el almacenamiento de información de una manera adecuada, para que la tabla pueda mostrar dicha información. Se puede ver un caso de uso en la figura 4.34.

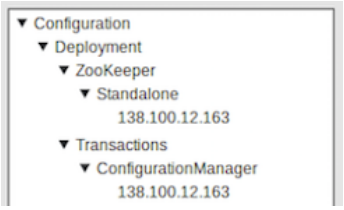

Figura 4.35: Ejemplo de Árbol en la aplicación

Árbol El árbol permite representar de una manera natural información jerárquica entre la que hay relación. Éste es uno de lo componentes finales principales de la aplicación, junto a las gráficas y las tablas. Es el núcleo de la vista del árbol de componentes del sistema a monitorizar. El árbol cuenta con muchas funcionalidades. De todas estas las que se emplean son las siguientes: primero, la capacidad de mostrar información ordenada de manera jerárquica; segundo, la emisión de eventos al pinchar sobre un elemento para la navegación por el árbol, y tercero como contenedor de una imagen actual similar a la del registro del modelo. Esta imagen es almacenada en un contenedor dedicado a ello, el cual es realmente un objeto. En él, la información se almacena de manera que las relaciones entre sus elementos quedan definidas. Se puede ver un caso de uso en la figura 4.35.

Como se ha visto, la mayoría de los componentes finales de la interfaz, cuentan con unos objetos asociados en los que se contiene la información que en ellos se muestra. Este es el caso del árbol, la tabla y las gráficas. Esta informacón es almacenada en estructuras de datos, que permiten un almacenamiento de la información, de una manera acorde a la forma en que se muestra esta por la interfaz.

4.4.2. Gráficas

En esta subsección se va a describir uno de los componentes finales claves en el desarrollo de la aplicación, las gráficas. En Vaadin, como se explicó en el capítulo 3, existe una librería de gráficas interactivas muy ricas en cuanto a visualización. Hay dos tipos, las gráficas normales y el timeline. Ambas permiten visualizar información numérica, y presentan una configuración muy flexible tanto en la forma que se muestran los datos, como en el estilo de estos, y la gráfica en general. Para la aplicación se ha empleado tan solo la parte de gráficas y no la del timeline, dado que el objetivo de la aplicación es mostrar

en tiempo real, la visualización de una métrica. Siendo el timeline quizás más adecuado para un histórico de los valores. Dentro de las gráficas disponibles (barras, columnas, áreas, etc), la que se ha utilizado ha sido el line chart o gráfica lineal. Esta consiste en la representación en dos dimensiones de un valor X e Y, en el cual la X representa a un valor temporal y el eje Y al valor de la métrica en las unidades correspondientes. A su vez cada uno de los puntos que se representan están unidos a su sucesor y a su punto posterior por una linea recta. Se puede ver un ejemplo de una gráfica en la figura 4.36.

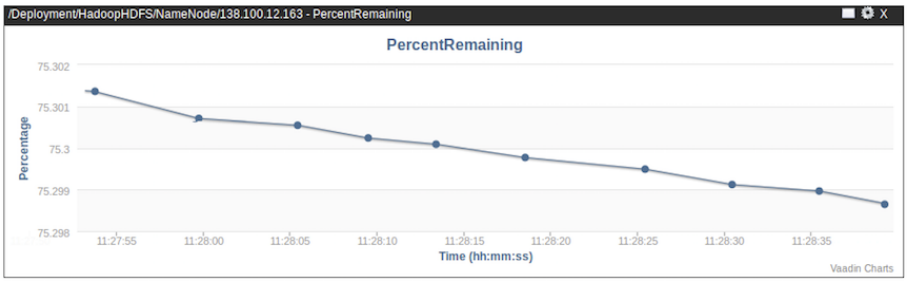

Figura 4.36: Gráfica de la aplicación

El manejo de las gráficas se puede dividir en dos apartados. La parte de configuración visual y la parte de los datos o información. A nivel de desarrollo existe un objeto gráfica que contiene el componente final, el cual tiene la referencia de otros dos objetos. Uno de ellos contiene a toda la configuración visual, y otro que contiene la información a mostrar.

Configuración

Como se puede ver en la figura 4.36, la imagen presenta una serie de estilos tanto en el exterior de los puntos como en los propios puntos. Decir que las gráficas cuentan con una configuración inicial estándar para todas ellas. Además, se le pueden realizar una serie de modificaciones de manera dinámica cuando una de estas gráficas está funcionando. Para la gráfica seleccionada se han realizado una serie de modificaciones iniciales, con respecto a la configuración por defecto. La primera de ellas es la eliminación de la leyenda. La segunda la definición del tamaño de los puntos y la linea, junto al color de esta. La tercera la modificación del color de las etiquetas del título y los ejes, al mismo color que los puntos. Además en el caso del eje X, se ha configurado para que el formato de los valores que se presenta sea en tipo fecha, mostrando hasta los segundos. La información de las etiquetas salvo la del eje X, es modificada en el momento de crear la vista de la gráfica según cierta información de la métrica. Esta información es el nombre y las unidades de la métrica. El marco que contiene a dicha gráfica también cuenta con información relativa a la métrica mostrada. Esta información refleja nuevamente el nombre de la métrica, junto al path del sistema del componente a monitorizar, que contiene a dicha métrica. En el marco también se aprecia, además de un botón para eliminar la gráfica y otro para hacerla más grande, uno de configuración. Este nos lleva a la vista siguiente. Fig 4.37.

Figura 4.37: Configurador de la gráfica.

Esta configuración permite la modficación de tres parámetros. El primero de ellos es el color del marco que contiene a la gráfica, siendo muy útil para la visualización en el dashboard de varias gráficas simultáneamente. El segundo es el número de puntos que se desean visualizar en la gráfica. El tercero cada cuanto tiempo se requiere que se obtenga un nuevo valor de la métrica.

Funcionamiento

Como ya se ha explicado, la mayoría de los componentes de Vaadin con cierta envergadura, disponen de un objeto a modo de contenedor de la información. Este objeto es requerido por el objeto de la gráfica para obtener de él los datos a mostrar. La gráfica por defecto cuenta con 30 puntos a visualizar, y se refresca cada segundo. Esta información puede ser modificada con el configurador. La gráfica, una vez que se inicia, cuenta con la información suficiente para conectarse al modelo, y obtener el último valor de la métrica requerida. Este proceso se hace repetidamente según el tiempo por defecto de 1 segundo, o el que se haya establecido para cada métrica. Los datos obtenidos desde el modelo son los formados por un tiempo y un valor. Éstos son almacenados en el objeto que se emplea de contenedor de la información, asociado a la vista. En caso de que se deseen visualizar menos puntos de los actuales, se eliminan puntos almacenados del objeto contenedor. Y en el caso contrario de que se requieran más, se hace más grande la ventana de visualización. Lo que equivale a almacenar más elementos en el contenedor. Básicamente, la cantidad de puntos almacenados en el objeto contenedor, es la cantidad de puntos que se visualiza en la aplicación. El objeto contenedor, en el caso de las gráficas, no es mas que una lista ordenada formada por objetos con dos valores, una X y una Y; o lo que es lo mismo un tiempo y un valor.
Todas las instancias de las gráficas llevan asociadas como cada vista de la aplicación un controlador. Este controlador es el que posee el conector con el modelo de la base de datos, y es el encargado de obtener la información. Además se encarga de almacenar los datos en el objeto contenedor asociado a la vista, y de modificar el tamaño de este contenedor, en caso de que a través de la configuración de la gráfica el usuario lo solicite.

4.4.3. Estilos

En el capítulo 3 se explicó todo lo relacionado con la parte de creación y modificación de temas. Vaadin cuenta con un muy buen diseño en cuanto a la separación de estos temas respecto de la lógica, y el diseño de la aplicación, permitiendo el desarrollo de la aplicación, y la edición de los temas de manera independiente. Estos temas están relacionados con las hojas de estilos de la aplicación, la modificación de los layouts de los componentes html, y con cualquier imagen que se desee añadir a la aplicación, como puede ser un logo.
Por defecto Vaadin incluye cinco temas, cada uno de los cuales ofrece unas hojas de estilos y un tipo de representación de los componentes html o componentes finales: como botones, etiquetas, etc. Esto temas son: valo, reindeer, chamaleon, runo y liferay. El que se ha empleado ha sido reindeer, por ser el más general y el que mejor se adapta a las modificaciones.
A nivel de desarrollo en la aplicación, en la clase principal se debe indicar que tema se requiere. A partir de ahí existen dos opciones para realizar las modificaciones del tema. La primera es crear una nueva entrada en el fichero de estilos, y añadir ésta al componente, especificándoselo en el código. La segunda opción, es en el fichero de estilos, heredar del estilo que esté empleando ese componte por defecto. Debido a que con la herencia se hereda toda la configuración por defecto, en la mayoría de los casos se ha optado por crear una entrada nueva y aplicar un nuevo estilo. Esta aplicación tan solo consiste en que en el código, se le especifique al objeto del componente final a modificar, cual es el nombre del estilo.
A tener en cuenta, que gran parte de la tarea con los temas es todo el posicionamiento de los layouts en la interfaz. En el caso de Vaadin ésta se realiza en el diseño de las propias vistas, a través de el editor visual de vistas. Figura 4.29. El posicionamiento por lo tanto es autogenerado en el código que se genera para las vistas, el cual es compilado y traducido a información válida para aplicarla a los estilos.

Modificación de layouts Los componentes html, por defecto tienen una disposición acorde a la especificada por el tema del reindeer. La mayoría de ellos se han mantenido por defecto. Se han modificado tan solo dos componentes. El primero, el componente final option, el cual muestra por defecto sus opciones en vertical. Este se cambió para que se mostraran en horizontal, como se puede ver en la figura 4.37. El segundo ha sido el posicionamiento del texto dentro de la mayoría de las etiquetas que se han empleado.

Estilos A la aplicación se han incluído una serie de estilos para hacerla más atractiva visualmente. Entre los colores empleados, los predominantes son: el negro (#000000), el blanco (#FFFFFF) y el gris (#E5E5E5). Mayoritariamente se han aplicado a los layouts principales como se puede ver en alguna de las figuras 4.14,4.16. También se han aplicado otro tipo de colores como: el azul (#52A8EA), el naranja (#EAA852) y el rojo (#EA5452), para los marcos de las gráficas o para mostrar en el menú de la aplicación en qué apartado se encuentra el usuario. Esto se puede ver en las mismas figuras anteriores. Los estilos también se han empleado para la especificación del tamaño del texto en las etiquetas, añadir bordes a layouts, añadir sombreados, etc. Otra característca interesante

de los estilos es la de que cuando se pasa por encima de un componte html, este lo detecta y cambia su estilo. Esto se ha aplicado sobre los botones del menú que se iluminan al posicionarse sobre ellos. A grandes rasgos, éstas han sido las modificaciones de estilos, con respecto al estilo por defecto del reindeer.

Imagenes En cuanto a las imágenes incluidas en la aplicación, tan solo se han utilizado dos. Estas son las que aparecen en la vista del marco representando a un botón para la configuración, y otro para aumentar la imagen. Se puede ver en la figura 4.30.

Capítulo 5

EJEMPLO DE USO: INTEGRACIÓN CON CUMULONIMBO

En este capítulo se va a describir un caso de uso para evaluar el sistema desarrollado. Para comenzar se va a realizar una introducción al sistema a monitorizar. Este es CumuloNimbo[1]. También se explicará como es descrito en el fichero de configuración del sistema de monitorización. A continuación se describirán las tecnologías empleadas como elementos del modelo: la base de datos y el registro. También se mencionarán los pasos para realizar el despliegue del monitor, a través del instalador creado para ello. Finalmente se describirá un caso de uso del sistema, a través de la monitorización de CumuloNimbo.

5.1. Introducción al sistema, subsistema y roles

El sistema sobre el que se va a realizar el caso de uso de la monitorizacón es CumuloNimbo. Este sistema se define como una Plataforma como Servicio (PaaS) de transacciones, multinivel y altamente escalable. CumuloNimbo es un proyecto Europeo financiado por la comisión Europea bajo la séptima edición del Programa Framework (FP7). La dirección y parte del desarrollo del proyecto se lleva a cabo desde el Laboratorio de sistemas Distribuídos[2] de la Escuela de Ingenieros Informáticos[3] de la UPM[4]. El proyecto se ha desarrollado entre el 1 de Octubre de 2009 y el 30 de Septiembre de 2013.
En la actualidad cada vez más y más software se está alojando en la nube, lo que es llamado software as a service (SaaS). Para el acceso a dicho software tan solo es necesaria una conexión a Internet, lo que lo hace altamente accesible. Pero de igual manera que antes, para realizar esta computación en la nube también son necesarias plataformas como bases de datos, etc. CumuloNimbo se ofrece como una plataforma como servicio (PaaS), para aplicaciones multi-capa que necesitan de un procesamiento transaccional ultra-escalable proporcionando el mismo nivel de consistencia y transparencia que un sistema de base de datos relacional tradicional. Los sistemas que se ofertan actualmente cuentan con la

[1] http://www.cumulonimbo.eu/
[2] http://lsd.ls.fi.upm.es/lsd
[3] http://www.fi.upm.es/
[4] http://www.upm.es/

problemática de que a medida que la aplicación escala, para que la base de datos escale, ésta se divide en fragmentos independientes que tan solo comparten el esquema de las base de datos, lo que propicia que muchas de las propiedades como son la atomicidad, consistencia, aislamiento y durabilidad (ACID) se pierdan. CumuloNimbo garantiza estas propiedades y lo hace gracias al diseño de su arquitectura y de los componentes que la forman. Además ofrece todas las bondades de SQL y soporta tecnología NoSQL [5].

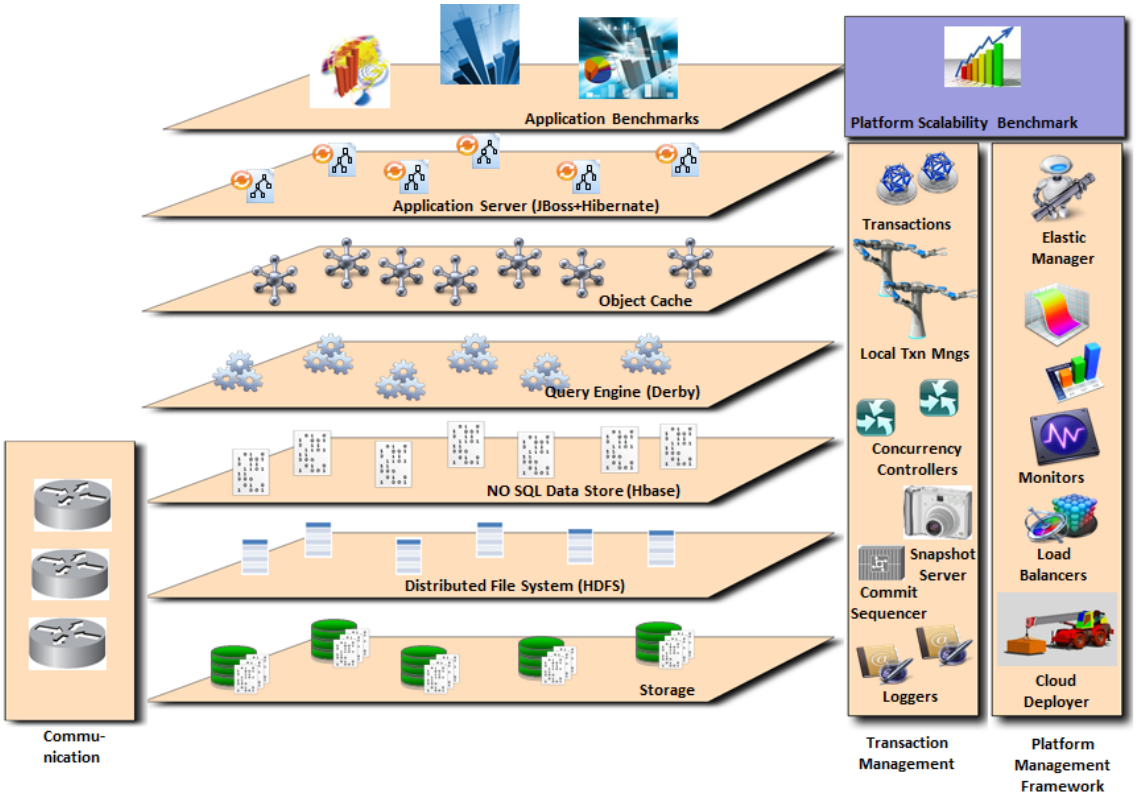

Figura 5.1: Subsistemas de CumuloNimbo [5]

En la figura 5.1 se muestran los subsistemas que componen CumuloNimbo. Como se puede observar, la arquitectura está formada por múltiples capas en las que en cada una se encuentra un tipo de tecnología con una funcionalidad específica. Mencionar que para conseguir el rendimiento deseado del sistema, cada nivel puede ser escalado añadiéndo más servidores. A continuación se describen cada una de las funciones de los distintos niveles [6].

- **Application Server**. Es la capa del servidor de aplicaciones. Es decir, la capa intermedia entre las aplicaciones y CumuloNimbo.
- **Object cache**. Es una caché distribuída de objetos. Dicha caché es compartida por todas las instancias del servidor de aplicaciones. Esto permite que cuando la aplicación solicite un objeto, este se buscará primero en dicha caché.
- **Query Engine**. Es la capa del motor de consultas SQL. A esta se accede cuando no se encuentra la información en la caché de objetos. El número de motores depende

del número de consultas complejas que son ejecutadas concurrentemente, y de la cantidad de peticiones concurrentes que un motor de consultas pueda manejar.

- **NoSQL data store**. El almacenamiento es NoSQL a través de un par clave-valor, en vez de un almacenamiento de una base de datos tradicional. La tecnología empleada es HBase[5].

- **Distributed file System**. HBase se sitúa encima de un sistema de ficheros paralelo-distribuido. Este se encarga de proporcionar el almacenamiento persistente de los datos. En esta capa se emplea Hadoop Distributed File System (HDFS[6]).

- **Transaction Management**. La gestión de las transacciones depende de un conjunto de componentes que son capaces de proporcionar además una alta escalabilidad. Estos son:

  - **Commit Sequencer**. Actúa como secuenciador de commits o compromisos.
  - **Snapshot Server**. Servidor de snapshot o instantáneas.
  - **Conflict Manager**. El manejador de conflictos.
  - **Loggers**

- **Platform Management Framework**. Es la capa encargada de tareas de gestión del sistema tales como el despliegue, la monitorización, equilibrado de carga dinámico y elasticidad. Para que esto sea posible, el sistema cuenta en cada capa con un monitor recogiendo datos sobre uso de recursos y mediciones de rendimiento.

Se ha visto a todos los subsitemas que componen CumuloNimbo. Decir que la mayoría de ellos cuentan con distintos roles en su interior como es el caso del subsistema *Transaction Management* que cuenta con los roles del *Commit Sequencer*, *Snapshot Server*, etc; o el caso del subsistema de *NoSQL dataStore* que cuenta con los roles del *Master* y los *Region Servers*. A su vez cada uno de estos roles cuenta con múltiples instancias para una ejecución real del sistema. Además cada una de estas instancias ejecuta en una máquina física diferente. La capa de comunicación que se ve en la figura 5.1 es la que permite la comunicaciíon entre todas ellas.
Como se ha dicho anteriormente, el sistema tiene la capacidad de que por cada capa de la que está compuesto, se exporten una serie de datos sobre el uso de los recursos del sistema y mediciones de rendimiento. Éstas, unidas a las mediciones que se obtienen de los componentes físicos del sistema a través de otras herramientas, permiten tener una visión total del sistema.

[5]https://hbase.apache.org/
[6]http://hadoop.apache.org/

5.1.1. Caso de CumuloNimbo a monitorizar y configuración de éste

La explicación que se ha hecho hasta ahora ha servido para describir a CumuloNimbo junto a todos sus componentes. El ejemplo de monitorización se va a llevar a cabo sobre un caso concreto del sistema, diseñado para evaluar una tarea específica; es decir, se va a preparar un caso de uso de CumuloNimbo, y a través del sistema de monitorización se verán los valores de rendimiento y uso de recursos, que exporta, y son recogidos por la monitorización.

El sistema va a ser desplegado en un cluster, debido al gran número de instancias por cada una de las capas de CumuloNimbo. Se van a especificar la cantidad de instancias por cada capa, y en qué máquina física se ejecutan. Además se describirá en qué consiste el fichero de configuración en el que se representa al sistema a monitorizar.

Descripción del sistema El sistema va a contar con la mayoría de los componentes de CumuloNimbo. A pesar de ello, algunos de ellos no van a proporcionar información relevante para este caso concreto. A continuación se enumeran los blades del cluster, especificando por cada uno de ellos, que roles de CumuloNimbo ejecutan. Decir que en un mismo componente físico puede haber dos intancias de dos roles diferentes de Cumulo.

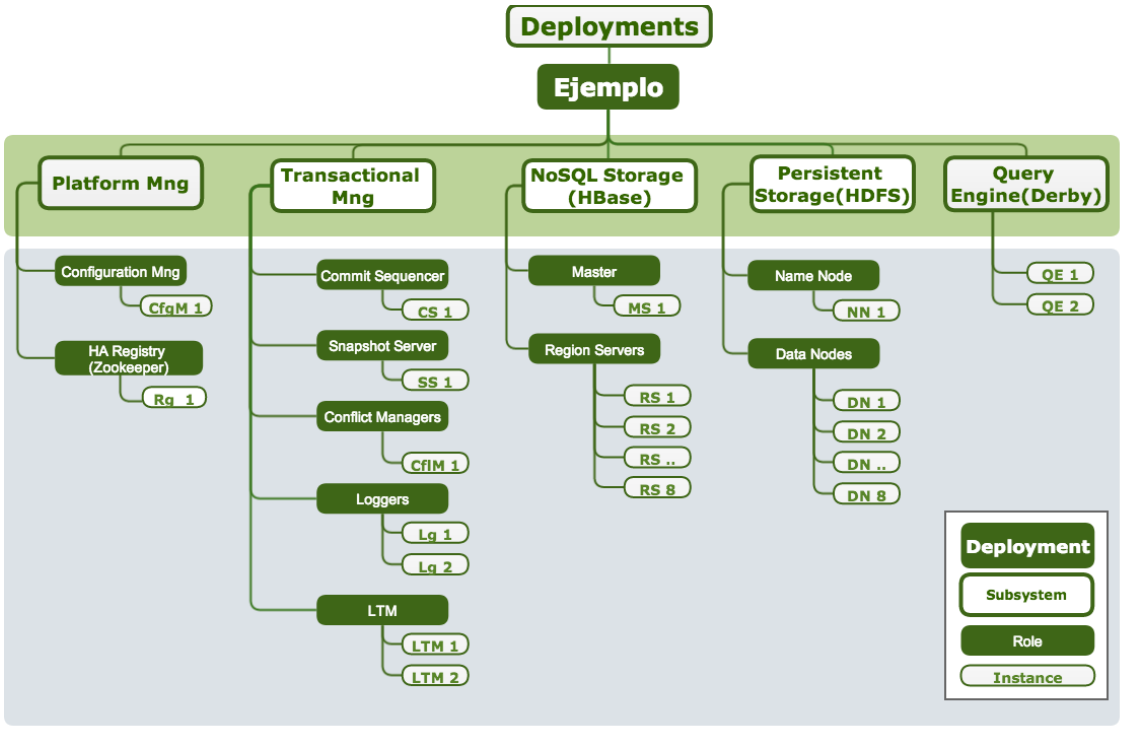

Figura 5.2: Diagrama de los subsistemas, roles e instancias de CumuloNimbo para el caso de uso

- **Blade 193**. Herramientas para el test de CumuloNimbo.
- **Blade 194**. CfgM-1 + CS-1 + SS-1
- **Blade 195**. CflM-1
- **Blade 196**. Herramientas para el test de CumuloNimbo.
- **Blade 201**. DN-1 + RS-1
- **Blade 202**. DN-2 + RS-2
- **Blade 204**. DN-3 + RS-3
- **Blade 205**. DN-4 + RS-4
- **Blade 206**. DN-5 + RS-5
- **Blade 207**. MS-1 + Rg-1 + QE-1 + Lg-1 + LTM-1
- **Blade 208**. DN-6 + RS-6
- **Blade 209**. DN-7 + RS-7
- **Blade 210**. NN-1 + QE-2 + Lg-2 + LTM-2
- **Blade 211**. DN-8 + RS-8

Se cuenta con un total de 14 máquinas físicas, y múltiples instancias de los roles agrupados en subsistemas de CumuloNimbo. A partir de ellos se obtendrán tanto métricas del sistema de los compoentes físicos (máquinas o blades), como de los componentes lógicos (subsistemas de CumuloNimbo e instancias). Esta descripción del sistema es la que consta en el fichero de configuración, para poder así ser monitorizada.

Fichero de configuración del sistema Como se ha dicho a lo largo de los capítulos anteriores, el sistema de monitorización cuenta con un fichero de configuración. Éste ha sido diseñado de manera que permita la representación de cualquier arquitectura de un sistema que se quiera monitorizar. De esta manera no se obliga como la mayoría de los sistemas de monitorización actuales, a ceñirte a sus niveles de representación, proporcionando así una visión del sistema que refleja totalmente la realidad. Se puede ver el fichero XSD de definición en la subsección 7.2.2 del anexo.
Como se puede ver en el fichero de definición, un despliegue de un sistema a monitorizar, se representa a través de una jerarquía (deployment) de recursos o componentes lógicos, que finalmente sus hojas terminan siendo instancias del sistema asociadas a un componente físico, y por una lista (installation) de estos componentes físicos. Es decir, un recurso lógico puede contener a otros recursos lógicos y físicos. Estos últimos no pueden contener a ningún recurso en su interior. Todo esto se traduce en que en el nodo de *installation* se listan todos los recursos o máquinas físicas en las que se ejecuta el sistema. Mientras que en el nodo de deployment, se describe la jerarquía lógica del sistema a monitorizar,

teniendo en sus hojas instancias del sistema asociados a máquinas físicas sobre las que ejecuta. En la figura 5.3 se puede ver un diagrama de la representación que proporciona el fichero. En esta representación se aprecia la relación entre las instancias de subsistemas que ejecutan en una máquina o componente físico, con la lista de los propios componentes físicos. Esta relación se hace a través de la dirección de red o IP. Esta diferenciación sirve para que en la aplicación se pueda tener una visión real del sistema, pero que a su vez en el *deployment* se ofrezcan métricas de la instancia, y en el *installation*, métricas de la máquina física.

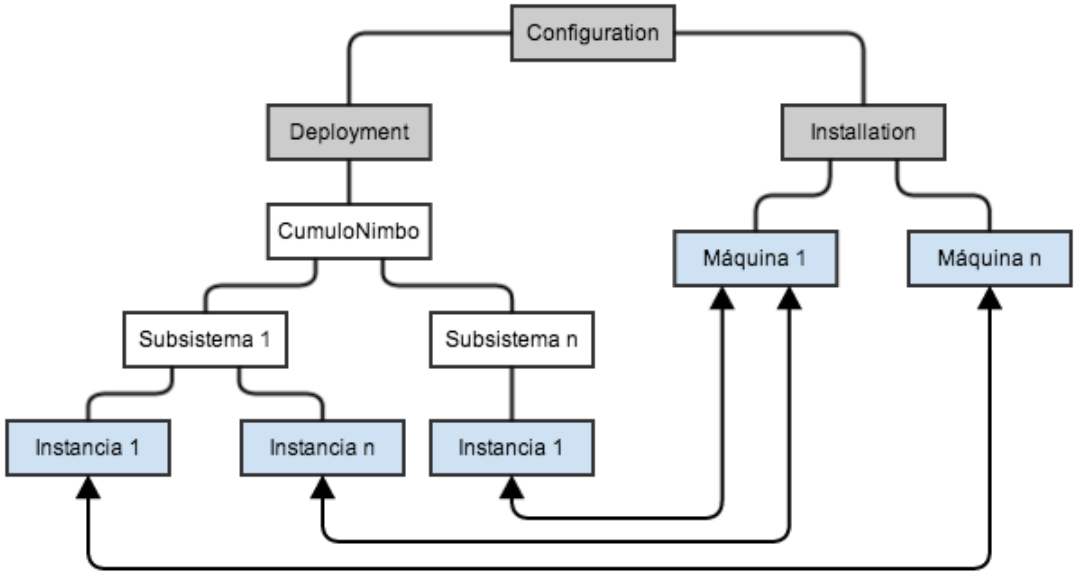

Figura 5.3: Elementos del fichero de configuración

En el fichero de configuración se encuentra toda la información relativa al sistema a monitorizar, necesaria para el funcionamiento del sistema de monitorización. Además de la información implícita de la jerarquía, también se dispone de la IP. Entre la demás información relevante se encuentran el *entityTypeName* y el *entityTypeVersion*. Ambos se emplean para la identificación de las métricas de cada instancia que se requieren monitorizar. Estos dos identificadores, hacen referencia a una serie de ficheros, en los que están definidas una lista de las métricas que se desean recolectar, para ser monitorizadas.
Finalmente, decir que el fichero de configuración permite la definición de la arquitectura de cualquier sistema, dividiéndo a ésta en *deployment* para la jerarquía lógica, y en *installation* para los nodos físicos. Siendo la relación entre estos a partir de la IP de las máquinas. Además el *entityTypeName* y el *entityTypeVersion* especifican que métricas se desean recolectar de cada una de las instancias de los subsistemas. Se puede ver el fichero de configuración XML empleado para este caso de uso en la subsección 7.2.1 del anexo. Este permite la visión de la arquitectura de Cumulonimbo dividida en subsistemas, roles e instancias, sin ningún tipo de problema.

5.2. Integración con la base de datos y el registro

A lo largo de la memoria se ha descrito el modelo del sistema de monitorización y su uso, pero en ningún momento se ha dicho qué tecnologías son empleadas para un caso de uso real. Decir que el diseño ha sido pensado para que tanto la tecnología de base de datos, como de registro, sea independiente del resto del sistema. Para ello se han definido unas interfaces, a través de las cuales, tanto el servidor de recolección de métricas como la propia aplicación web, puedan interactuar con dicho modelo. Esto permite que múltiples tecnologías puedan ser empleadas como modelo, simplemente si son capaces de implementar la interfaz propuesta.
Recordar que en el sistema, el modelo tiene dos funciones. La primera de ellas la realiza el registro de alta disponibildiad, el cual cuenta en todo momento con una imagen actualizada del sistema a monitorizar. La segunda la realiza la base de datos siendo contenedora de toda la información que se recolecta del sistema a monitorizar. En este caso de uso, las tecnologías usadas como registro y como base de datos son Zookeeper[7] y HBase[8] respectivamente.
Nota. El desarrollo del modelo no es de autoría propia. Es decir, los conectores, las interfaces, las implementaciones para Zookeeper y HBase, etc. Este proyecto tan solo hace uso de los conectores para acceder a los dos elementos del modelo.

5.2.1. HBase

Hoy en día, cada vez se le da mas importancia a determinada información, que hace poco tiempo se desechaba por falta de capacidad de almacenamiento y de procesamiento. Para manejar y procesar dicha información se necesita tecnología que pueda albergar grandes cantidades de datos, y que la lectura y escritura de éstos sea muy eficiente. HBase es una base de datos no relacional (NoSQL) y distribuída. Esta se ejecuta sobre un sistema de ficheros distribuído que es Haadop Distributed FileSystem[9] (HDFS). De esta forma HBase permite un acceso altamente tolerante a fallos, y la capacidad de almacenar y leer altas cantidades de datos, en tiempos muy reducidos. Toda la información se almacena en grandes tablas, formadas por filas y columnas, que contienen pares clave-valor. Por su tolerancia a fallos, ser distribuída, y ofrecer un alto rendimiento de lectura y escritura, HBase ha sido elegido como gestor de base de datos para el sistema de monitorización. Además de que la información que se almacena, principalmente son pares clave valor, en los que la clave es el timestamp de una métrica, y el valor es el dato relativo a dicha métrica [1].

5.2.2. Zookeeper

Tradicionalmente los programas de ordenador ejecutan en una sola máquina, siendo independientes del resto del mundo. Hoy en día con el Big Data y el Cloud Computing, las aplicaciones están formadas por múltiples programas independientes, que interactúan

[7] http://zookeeper.apache.org/
[8] https://hbase.apache.org/
[9] http://hadoop.apache.org/

unos con otros para alcanzar un objetivo. La solución tradicional para su coordinación ha sido la ardua tarea de implementar mecanismos de comunicación entre ellos, que en la mayoría de las ocasiones no lleva al éxito, y provoca que los posibles puntos de fallos sean muy grandes. Zookeeper se presenta como un elemento de coordinación, en el que en caso de fallo, su localización es mucho más simple. Por coordinación se entiende que es un servicio centralizado que proporciona la persistencia de información como configuraciones, servicios de grupo, espacio de nombres y sincronización distribuída. Además el servicio es si mismo es distribuído y altamente fiable. Por todas estas propiedades, Zookeeper ha sido elegido para ser la tecnología empleada como registro. También por que su almacenamiento en un árbol de directorios, es adecuado para almacenar información de la jerarquía del sistema a monitorizar, de una manera natural. La jerarquía que se ha visto antes en la subsección 5.1.1 es almacenada como un árbol de directorios en Zookeeper. Además, en cada uno de los nodos en los que es necesario, se almacena información de la IP, y el nombre y la versión del entityType.
Cuando la aplicación crea el controlador para el acceso al registro, se subscribe como listener para ciertos eventos que el registro genera. Entre ellos están el aviso en tiempo de ejecución de la adicción o eliminación de un nodo, o de una rama entera del árbol de directorios que Zookeeper gestiona [7].

5.3. Despliegue automatizado

Como en cualquier proyecto de cierta envergadura, el uso de sistemas de control de versiones es obligado. Además cuando un proyecto necesita de otros para funcionar, un gestor de dependencias es lo más viable para que no se presenten problemas, y no emplear tiempo en tareas que no sean el desarrollo. En el caso de este proyecto, todo el desarrollo del sistema de monitorización ha sido versionado, y sus diferentes dependencias gestionadas. Se ha empleado como gestor de versiones Git[10], y dado que se ha desarrollado en el lenguaje Java, se ha empleado como gestor de dependencias Maven[11].
El proyecto del sistema de monitorización está divido en 3 subproyectos. Estos son el agente, el servidor y la aplicación web. El proyecto padre que los contiene es el que está versionado. El funcionamiento con las versiones ha sido el de llevar a cabo el desarrollo en ramas independientes, que confluyen en la rama master, en el momento de realizar integración entre los distintos subproyectos, ya que cada uno es desarrollado de manera independiente.
Las dependencias entre los subproyectos y las de alguno de ellos con otros externos, ha llevado a que se realice su gestión con Maven. Por ejemplo en el caso de la aplicación, esta tiene dependencias con el proyecto del servidor y el del agente. Maven también se ha empleado para realizar la compilación de los proyectos, y el despliegue en un servidor de aplicaciones para el caso de la aplicación web. Llegados a un cierto punto del desarrollo, era necesario que para probar el sistema se pudiera de alguna forma tener un ejecutable, en el cual estuvieran contenidos los tres subproyectos listos para funcionar, con todas sus dependencias internas y externas. Para dar solución a ello se creó un instalador o starter. Maven ofrecía esta posibilidad. Se creó un nuevo proyecto, el cual tenía como dependencias a todos los proyectos del sistema de monitorización, y a proyectos externos. Además éste contiene los ficheros de configuración del servidor, del agente, el fichero de configuración del sistema, los ficheros que contienen las métricas para cada componte de CumuloNimbo, etc. A partir de esta información, y con la correcta adecuación del fichero de configuración de Maven a las necesidades, se genera el instalador. Este instalador que se genera contiene:

- Fichero de ejecución para el servidor y el agente.
- Carpeta con el servidor Apache Tomcat[12], que por defecto ya contiene a la aplicación web en su interior, para que ésta se despliegue nada mas se inicie.
- Ficheros estándar de configuración para el servidor y el agente. Éstos deben ser modificados para el caso de uso.
- Fichero de configuración estándar para la configuración del sistema a monitorizar. Este debe de ser modificado para el caso de uso.
- Ficheros con la definición de las métricas para el sistema CumuloNimbo.

[10] http://git-scm.com/
[11] http://maven.apache.org/
[12] http://tomcat.apache.org/

A partir de los recursos generados por el instalador, y con la debida configuración de los ficheros del servidor, agente y sistema, se puede desplegar el sistema de configuración. Explicación ,ás detallada de esto en el anexo 7.1.
Nota. El instalador del sistema de monitorización es de autoría propia.

5.4. Caso de ejecución

A continuación se va describir el caso de uso que se ha venido introduciendo en las secciones anteriores de este capítulo. En este se describirán los pasos realizados en la aplicación web, y se visualizarán métricas de los componentes físicos y lógicos de CumuloNimbo. Los elementos principales del caso de uso son el lugar de despliegue (los blades del LSD), el sistema de monitorización y el caso de CumuloNimbo descrito.

5.4.1. Inicio de los sistemas

El inicio de CumuloNimbo está fuera del alcance del proyecto. Tan solo decir que el caso de uso de CumuloNimbo que se ha evaluado es el que se ha descrito en la subsección 5.1.1. Para el inicio del sistema de monitorización se ha empleado el instalador descrito en la sección 5.3. Este instalador cuenta con tres ficheros que deben de ser configurados: el fichero de configuración del servidor, el del agente y el del sistema a monitorizar. El último de estos ya fué descrito anteriormente en la subsección 5.1.1. El fichero de configuración de CumuloNimbo que se ha empleado para este despliegue, se muestra en el anexo 7.2.1. En la sección 7.1 del anexo se ha escrito un manual con la explicación de como configurar los ficheros del agente y el servidor, además del general para configurar al sistema a monitorizar. En este manual también se explica como iniciar el servidor, la aplicación y los agentes. Finalmente decir que para la monitorización, no se han empleado las mismas máquinas que se han utilizado para el despliegue de CumuloNimbo. Obviamente para los agentes sí , ya que se ejecutan en estas máquinas. No lo hacen así la aplicación, el servidor y los componentes del modelo.

5.4.2. Descripción del caso

Una vez que la aplicación está desplegada, la primera imagen que se visualiza es la mostrada en la figura 5.4. En ella se ve un formulario que debe de ser rellenado, para indicarle a la aplicación en donde se encuentra el modelo del sistema de monitorización, y qué sistema debe de monitorizar. La información solicitada se detalla a continuación, junto a los valores introducidos:

- **RootPath**: /monitoring, path del registro a partir del cual se encuentra la información del sistema a monitorizar.
- **RegistryIP**: 192.168.1.191, dirección IP en la que se encuentra el modelo.
- **RegistryPort**: 2181, puerto en el que está escuchando el registro.
- **RegistrySessionTimeOut**: 10000, tiempo para la expiración de la sesión con el registro.

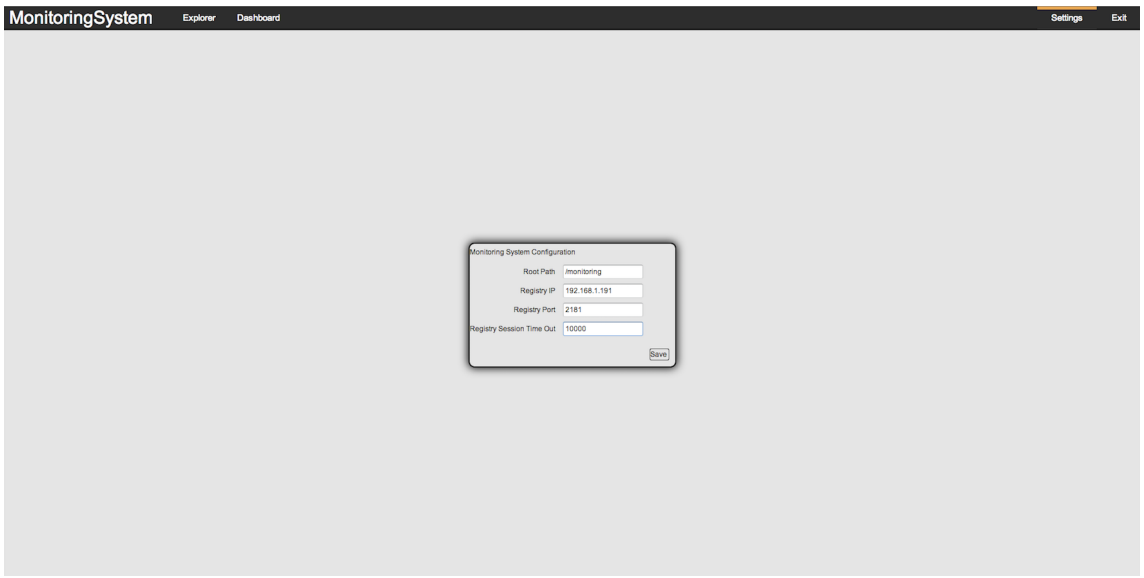

Figura 5.4: Pantalla de inicio de la aplicación

Una vez que la aplicación procesa la información anterior, lo primero que hace es obtener del registro toda la información necesaria sobre el sistema a monitorizar. Esta está contenida en un árbol de directorios dentro del registro, que es replicado por la aplicación, y mostrado en forma de árbol al usuario. En la figura 5.5 se puede ver el árbol que representa a la arquitectura de CumuloNimbo que se está monitorizando en este ejemplo, como se describió en la figura 5.2.
En el árbol se visualizan todos los subsistemas y roles de CumuloNimbo. Decir también que para este caso concreto se visualizan también dos herramientas empleadas para la evaluación de Cumulo, Escada y EscadaOLAP. Simplemente mencionarlas, pero no se hará nada con ellas en esta descripción del caso de uso.
Una vez con el árbol de la arquitectura cargado en la aplicación, el usuario puede comenzar a navegar por la arquitectura de CumuloNimbo, y a consultar sus métricas. El sistema proporciona métricas agregadas de cada una de las instancias de un rol, o métricas individuales de cada instancia. A continuación se van a mostrar las métricas agregadas para un rol del sistema, y después se mostrarán de manera individual para una instancia. Cuando el usuario selecciona desde el árbol un elemento, en este caso un rol, la vista que se muestra es la de la figura 5.6. La vista del explorador para cada uno de los elementos del sistema siempre es la misma. Esta está formada por: una tabla en la parte superior con información sobre ese elemento, una tabla intermedia con los hijos de ese elemento y a través de la cual también se puede navegar, y una tabla inferior que muestra las métricas de ese elemento, en caso de que las tenga.
Para la figura 5.6, al tratarse de un rol, la tabla superior muestra propiedades de ese elemento, la tabla intermedia las instancias de ese rol, y la tabla inferior las métricas agregadas de todas sus instancias. Si en la tabla intermedia se selecciona alguna de las instancias, se muestra la vista del explorador para dicha instancia, figura 5.9.

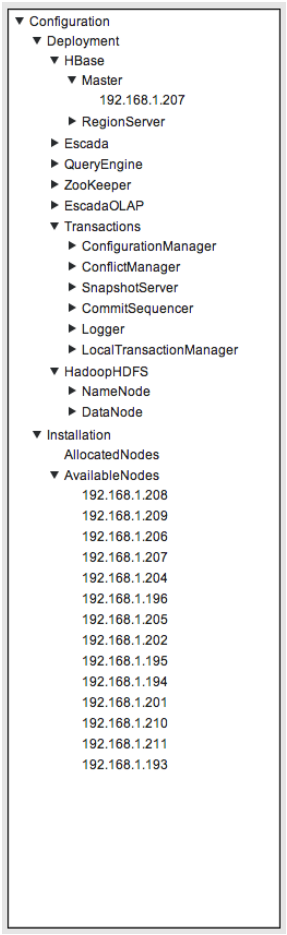

Figura 5.5: Árbol con la arquitectura de CumuloNimbo

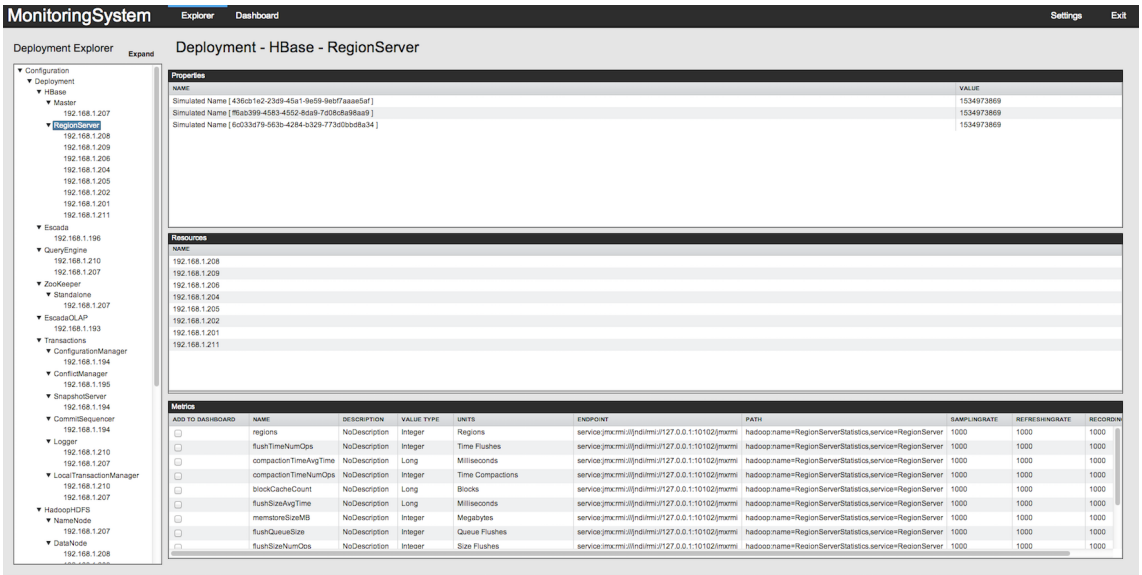

Figura 5.6: Vista del explorador del sistema para un rol

En el caso de la tabla de métricas, hay dos posibilidades: agregar la métrica al dashboard, figura 5.13, o seleccionar la métrica para visualizarla. Si se selecciona la métrica para visualizarla, la imagen que se muestra es la de la figura 5.7.

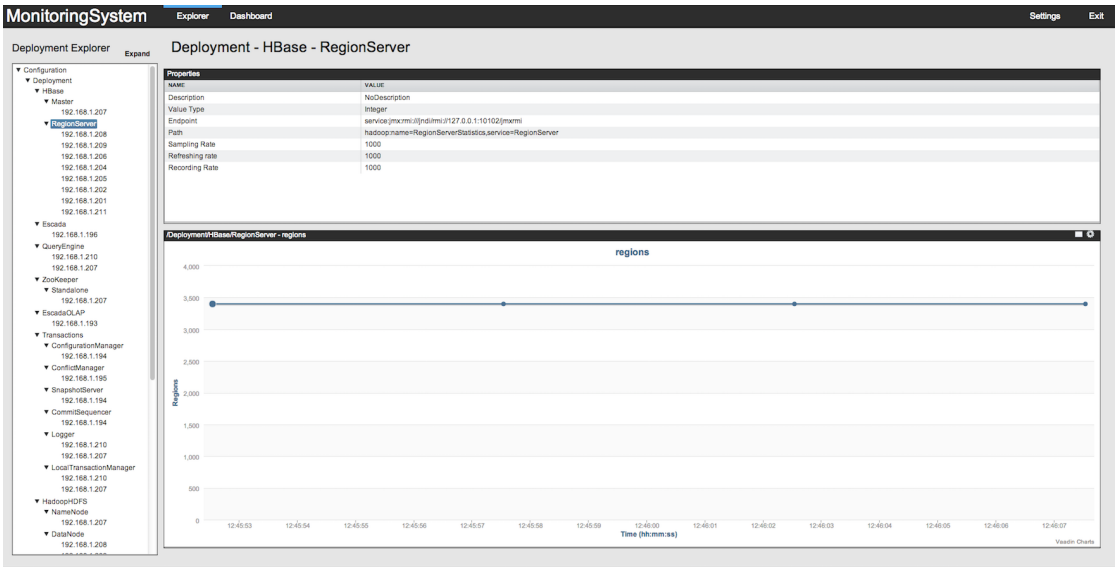

Figura 5.7: Vista de una métrica de un rol

Esta vista está compuesta por una tabla superior con propiedades de dicha métrica, y una gráfica con la monitorización de esta. Como se ha dicho a lo largo de la memoria, las gráficas son configurables. Esta configuración permite: seleccionar el color del marco de la gráfica, seleccionar la velocidad de muestreo de la gráfica, y seleccionar el número de puntos a visualizar en la gráfica. Una imagen de dicha configuración se puede ver en la figura 4.37.

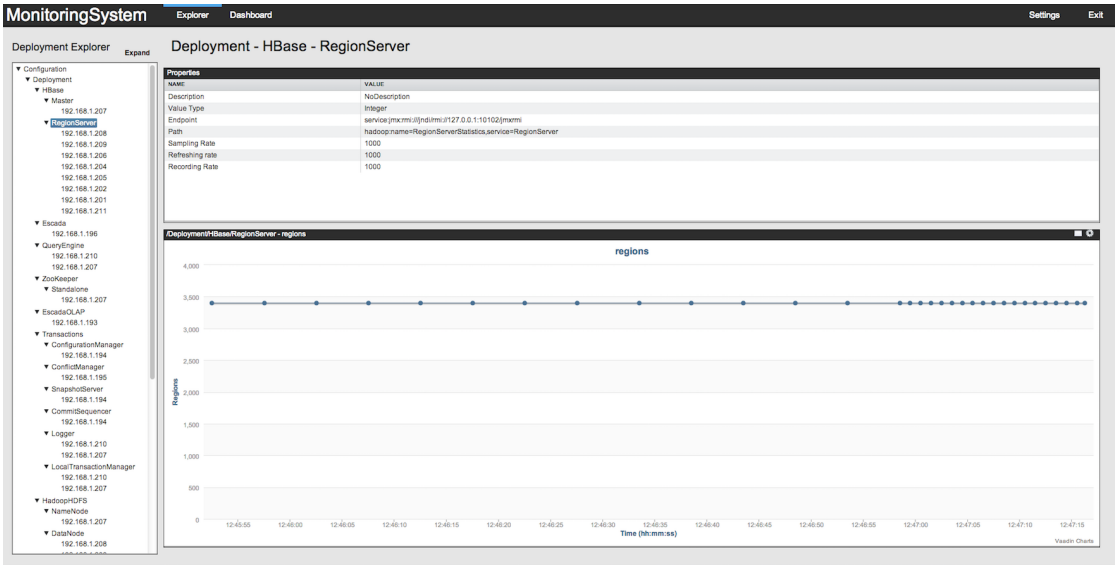

Figura 5.8: Vista de una métrica de un rol, modificada

Si por ejemplo configuramos la gráfica 5.7, y le incrementamos la velocidad de muestreo, el resultado es el que se muestra en la figura 5.8.

Ahora se va a visualizar un elemento hoja o instancia de la arquitectura. La imagen del explorador se puede ver en la figura 5.9. Esta es similar a la vista del elemento superior, salvo que en la ventana de los recursos hijos, obviamente se encuentra vacía. En la vista de las métricas, éstas son las mismas que para el nodo superior. A partir de ellas, junto a las de todos sus hermanos, es con las que se generan las del nodo padre, llamadas agregadas.

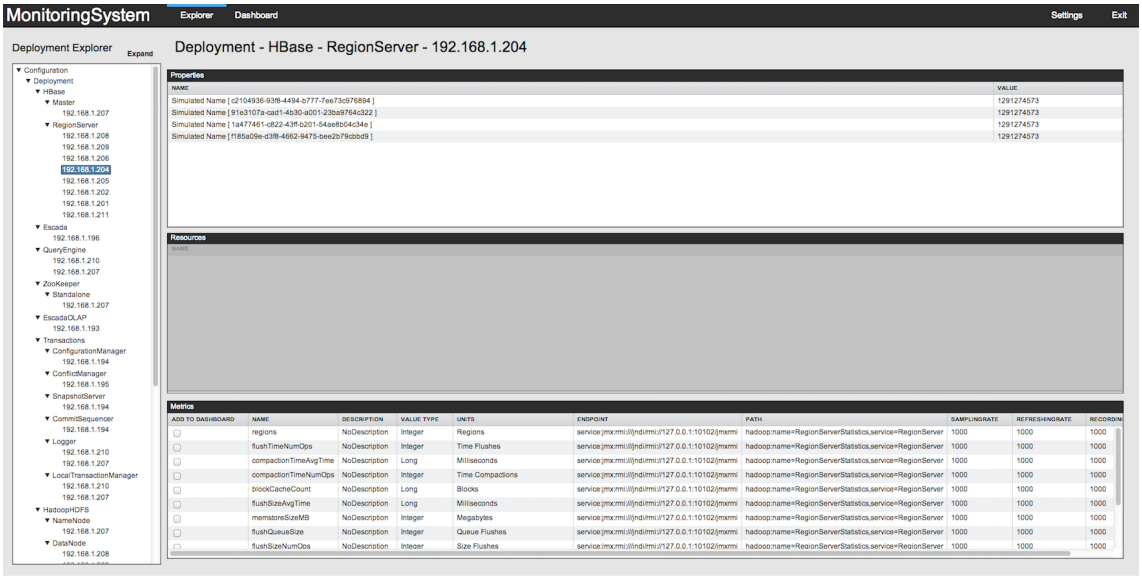

Figura 5.9: Vista del explorador para una instancia de un rol

Esto se puede comprobar visualizando una métrica, por ejemplo si seleccionamos la misma que en el padre (*regions*), en este tenía un valor aproximado de unos 3500, dado que hay ocho instancias de dicho rol, y si suponemos que son equivalentes, la métrica (*regions*) debería de ser aproximadamente de unas cuatrocientas regiones. Como se puede comprobar en la figura 5.10, esto es así. La vista de la métrica, como se puede comprobar, es la misma que para el caso anterior y para cualquier métrica vista desde el explorador.

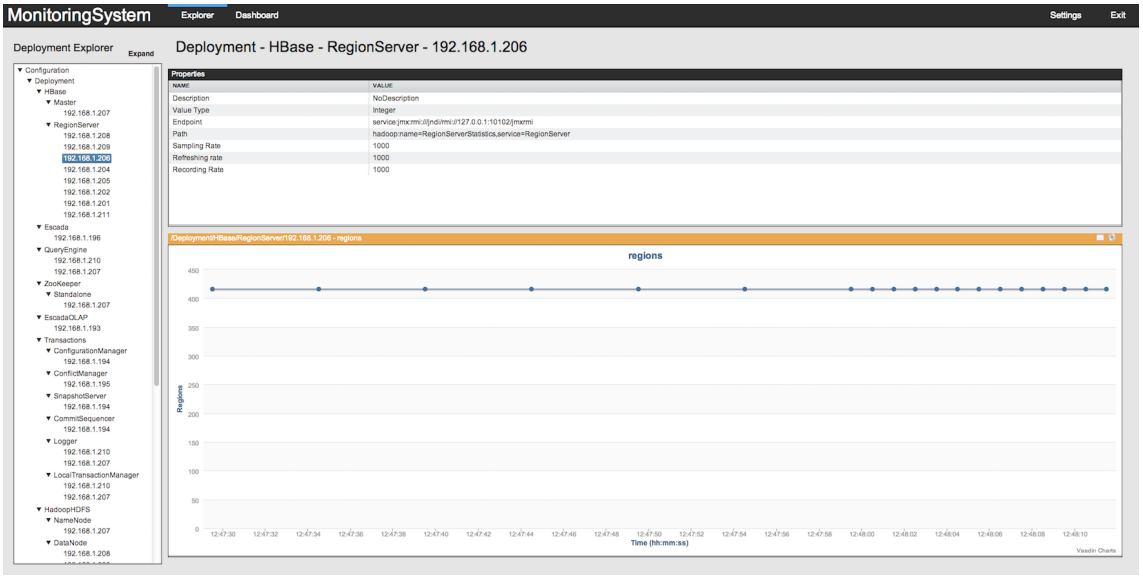

Figura 5.10: Vista de una métrica de una instancia

Una de las características del sistema es la capacidad de mostrar métricas de los componentes lógicos de la arquitectura, pero también de los físicos. Cada una de las instancias de los roles, son ejecutadas sobre un nodo físico que se diferencia por su IP. En el árbol del sistema 5.5 se puede ver que la visión de la arquitectura que se proporciona, muestra tanto la jerarquía lógica, como los nodos físicos. Cada uno de estos nodos se relaciona con las instancias de su rol a través de la IP. La visión del explorador para los nodos físicos es exactamente la misma que para la jerarquía lógica, figura 5.11. A su vez, estos también cuentan con agregación de sus métricas para tener una visión global del sistema físico.

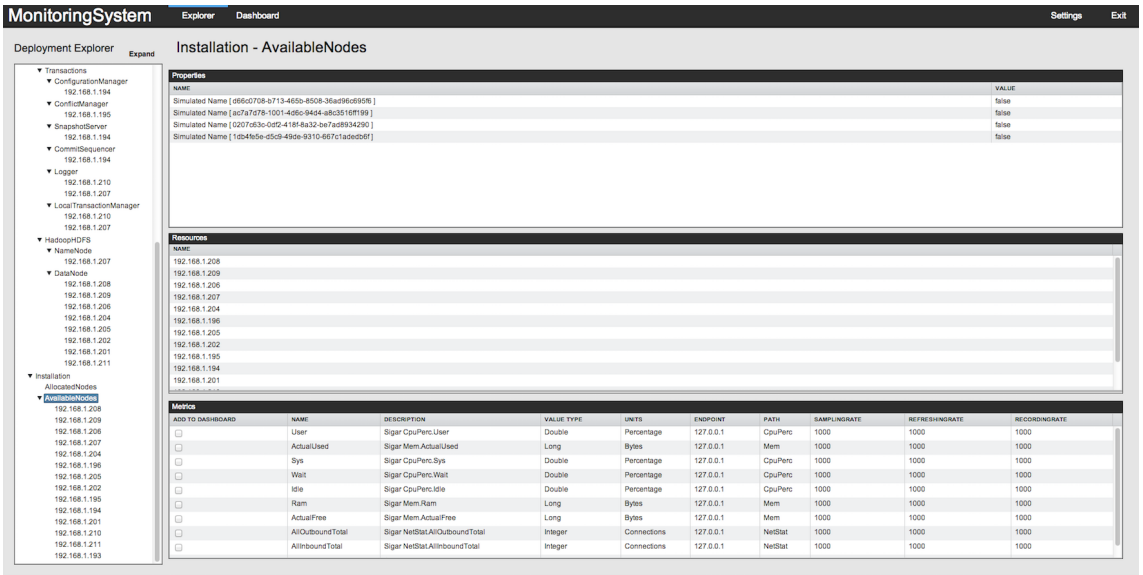

Figura 5.11: Vista del explorador para la agregación de nodos físicos

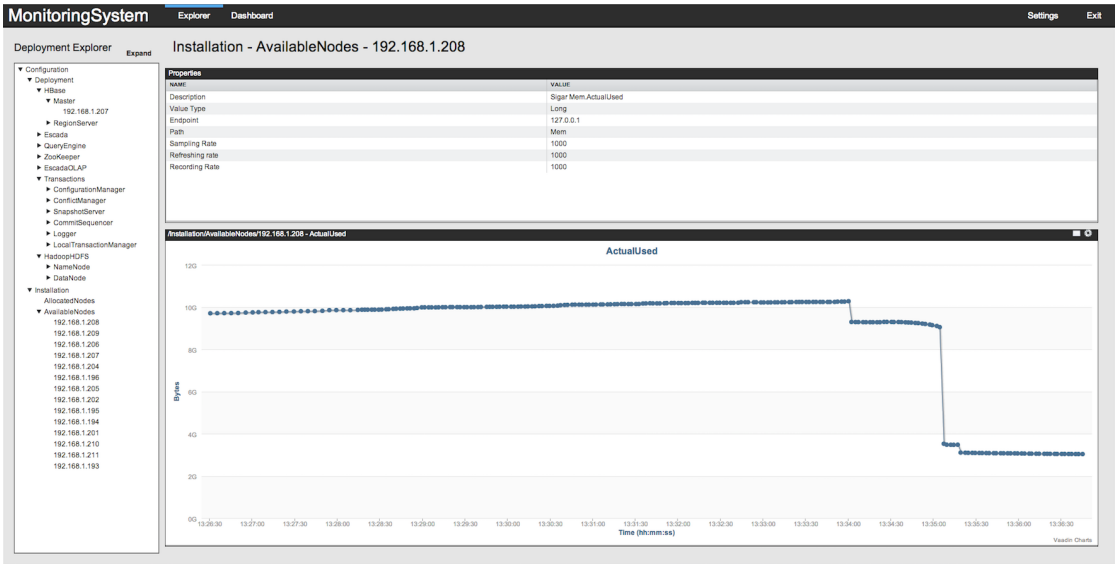

Figura 5.12: Vista de una métrica agregada de un nodo físico

Como se puede comprobar en el explorador, en la vista de los hijos, se ven todas las IP's de cada una de las máquinas físicas del sistema a monitorizar. Para los nodos físicos, la vista de las métricas también es la misma. En la figura 5.12, se puede ver una imagen de una métrica agregada de todos los nodos físicos del sistema.
Una vez que se han visto las posibilidades del explorador, se va a describir el dashboard. Decir que para añadir una métrica al dashboard, ésta debe de ser seleccionada desde la tabla de métricas del componente al que pertenece. Todas las tablas que muestran métricas en la parte izquierda cuentan con un checkbox, que si se marca, propicia que la métrica sea mostrada en el panel del dashboard. Se puede ver este checkbox en la figura 4.34 que muestra una tabla de métricas.
Por lo tanto, una vez que se han añadido cuatro métricas de algunos de los componentes del sistema, se selecciona la vista del dashboard. Esta se puede ver en la figura 5.13. Como se puede observar, el tipo de gráfica es el mismo que el mostrado en el explorador, solo que ahora se permite visualizar varias métricas de forma simultánea.

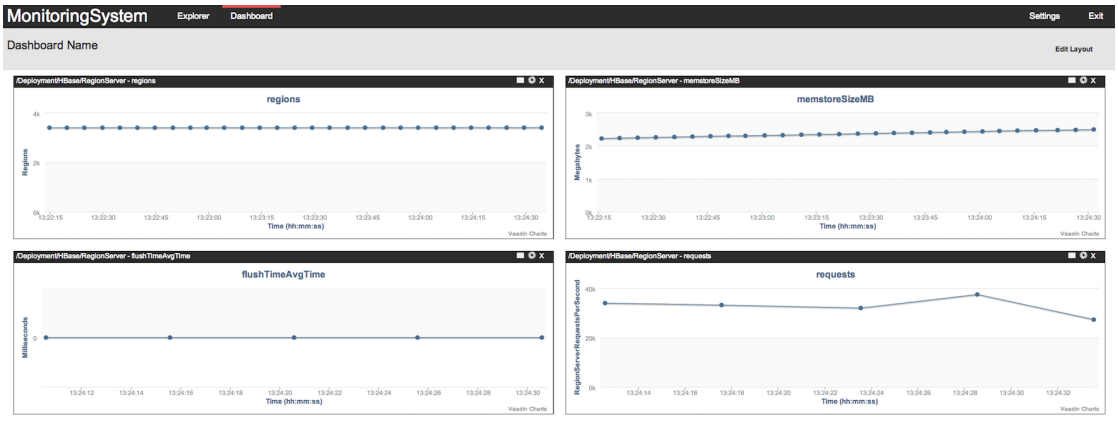

Figura 5.13: Vista del dashboard con cuatro métricas añadidas

Al igual que en la vista del explorador, estas gráficas también se puede configurar. El resultado se puede ver en la figura 5.14.

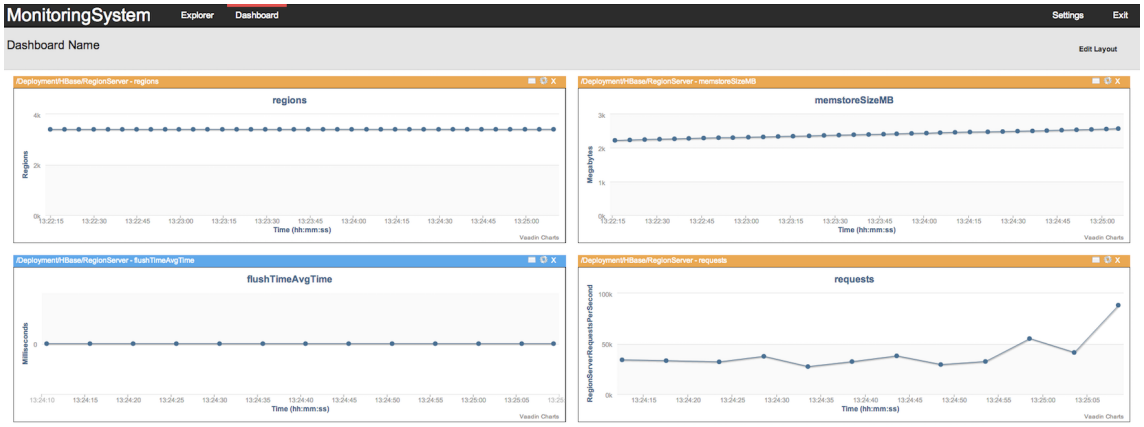

Figura 5.14: Vista del dashboard con cuatro métricas añadidas y modificadas

El dashboard permite visualizar las gráficas de dos maneras, desplegándolas en dos columnas o en tres. Por defecto la configuración las añade en dos columnas. En la vista del dashboard, arriba a la derecha se dispone de un botón, que al pulsarlo, muestra un panel emergente para seleccionar el tipo de vista que se quiere. Se puede ver una imagen en la figura 5.15.

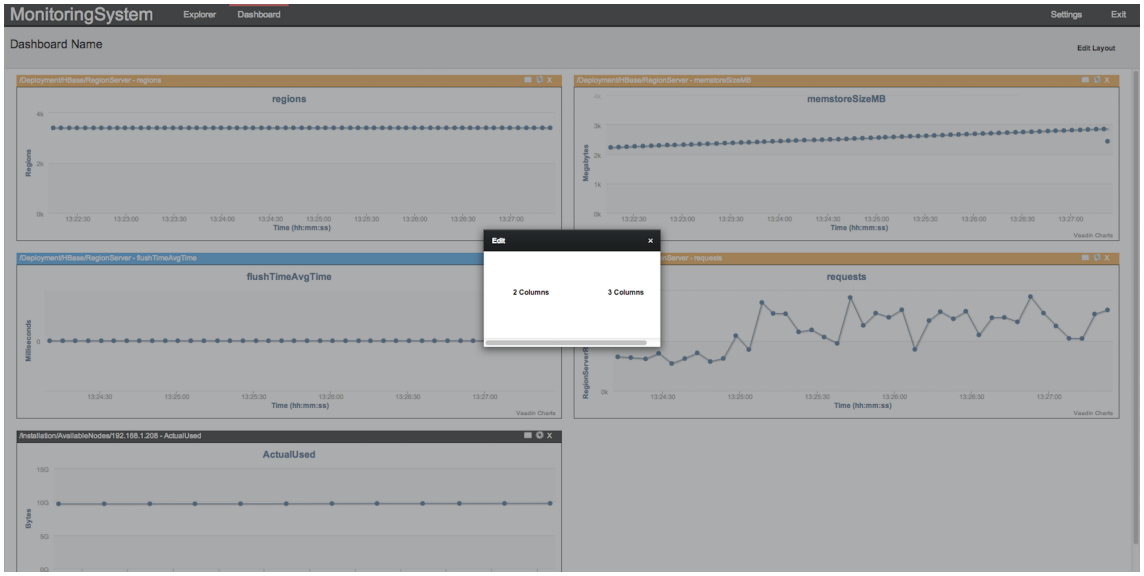

Figura 5.15: Vista del configurador de la vista del dashboard

Una vez que se ha seleccionado la vista, en este caso tres columnas, la vista reorganiza las gráficas en el mismo orden en el que estaban añadidas. El resultado final se muestra en la figura 5.16.

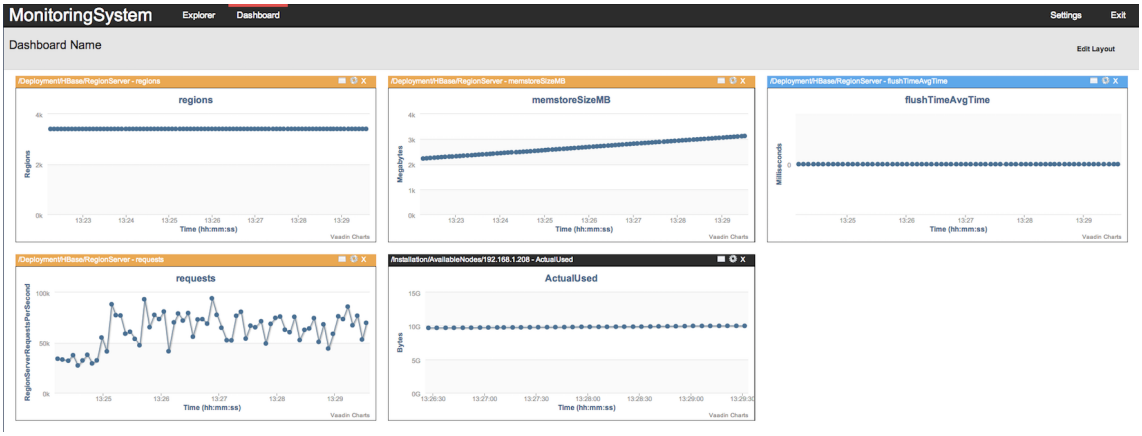

Figura 5.16: Vista del dashboard configurada con tres columnas

Como cabe esperar, el dashboard cuenta con la posibilidad de eliminar gráficas. Para ello tan solo es necesario pulsar el botón de eliminar en el marco de la gráfica, ésta es eliminada del dashboard, y el checkbox de la tabla donde se seleccionó es reiniciado. Se puede ver en la figura 5.17 una imagen del dashboard después de haber eliminado tres de las cinco métricas que se estaban monitorizando, y haber vuelo a poner la vista en dos coluumnas.

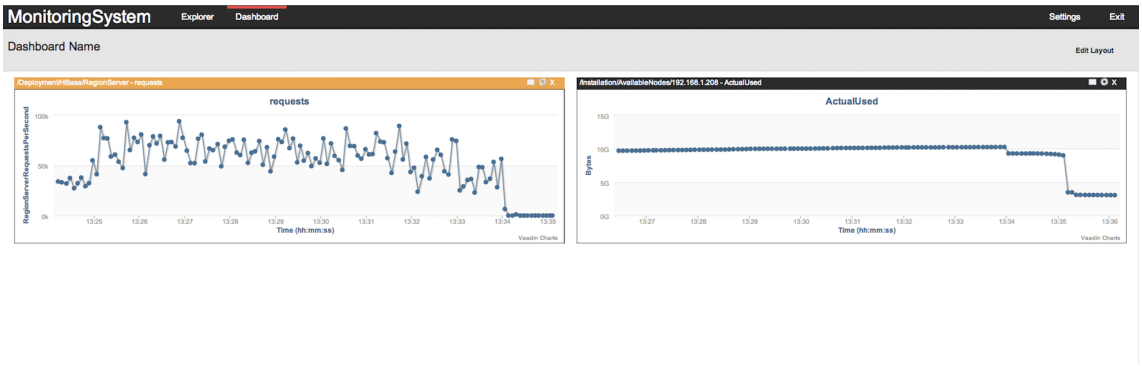

Figura 5.17: Vista del dashboard después de eliminar tres gráficas de cinco

Desde cualquier lugar de la aplicaicón, el usuario puede cerrar sesión pulsando el botón (exit) en la parte superior derecha del menú de la aplicación. Esto hace que la aplicación cierre la sesión y que vuelva a mostrar el panel de configuración (figura 5.4), en caso de que el usuario se quiera conectar a la monitorización de un sistema.

Con esta descripción se ha dado una visión de todas las funcionalidades de la aplicación web, a la vez que se ha visto como ésta soporta la monitorización de CumuloNimbo.

Capítulo 6

CONCLUSIONES Y FUTURAS LÍNEAS DE TRABAJO

El trabajo realizado a lo largo de este proyecto fin de grado ha permitido crear una aplicación web integrada a la perfección con el resto del sistema de monitorización. Éste ha sido testado satisfactoriamente sobre la plataforma CumuloNimbo. También se ha cumplido el principal objetivo, el ofrecer al usuario una visión real de la arquitectura del sistema que se está monitorizando. A pesar de todo ello, se han identificado una serie de puntos de mejora, con respecto al sistema en general, y en especial a la aplicación web.

6.1. Conclusiones generales

El desarrollo del proyecto se ha centrado principalmente en la aplicación web, aunque también se han realizado tareas de integración y de definición de la consistencia del sistema. Ésta se ha desarrollado con el framework Vaadin, el cual en mi opinión ha sido un total acierto. El aprendizaje de Vaadin ha sido muy rápido, lo que ha permitido que en muy poco tiempo, hubiera un prototipo funcional con el que empezar a integrarse con el resto del sistema. A su vez, la librería tanto de gráficas como en general las posibilidades de configuración en Vaadin, permiten que básicamente el desarrollador tan solo tenga que centrarse en la parte de la lógica del sistema, no teniéndose que preocupar por la parte visual, nada más que para crear las vistas y configurarlas. El único pero del framework, quizás es el rendimiento que ofrece. Se ha comprobado que cuando al dashboard se añaden por ejemplo unas diez gráficas con una velocidad de refresco de 1 segundo, el rendimiento del sistema se ve afectado. Esto es quizás debido a su arquitectura cliente-servidor, la cual beneficia en la seguridad, pero quizás la continua comunicación hace que su rendimiento baje.
Al inicio del proyecto, el objetivo principal era el de realizar un sistema de monitorización que permitiera ver de una manera clara la arquitectura del sistema a monitorizar, y que éste fuera independiente de cualquier sistema. A pesar de que tan solo se ha probado el sistema con CumuloNimbo, tengo la total seguridad, de que si un sistema se adaptara a la configuración de la monitorización, ésta sería capaz de llevarla a cabo. Con respecto a los objetivos de la aplicación, éstos han sido alcanzados. Los dos objetivos básicos eran

el desarrollo de un explorador para navegar por la arquitectura del sistema y visualizar sus métricas, y un dashboard en el que poder mostrar cualquier métrica de las vistas en el explorador. Como se ha dicho, éstos dos objetivos han sido cumplidos.
En el capítulo anterior se ha descrito el caso de uso sobre la plataforma CumuloNimbo, el cual ha sido totalmente satisfactorio. Decir que ésta no era la primera vez que se testaba el sistema sobre la plataforma, éste ya había sido probado muchas más veces, sobre todo para realizar pruebas de integración. Esto también permitió que con el paso de las ejecuciones, se fueran mejorando ciertas caracteríStcias de la aplicación, sobre todo a nivel de la interfaz. También sirvió para identificar muchas mejoras para implementar, las cuales estaban fuera del alcance de dicho proyecto, y que se comentarán en el apartado de futuras líneas de desarrollo.
Decir que el sistema que se ha desarrollado, está siendo empleado por CumuloNimbo para hacer demostraciones de su sistema.

6.2. Futuras líneas de desarrollo

Las futuras líneas de desarrollo se dividen en mejoras necesarias en todo el sistema de monitorización, y mejoras tan solo a nivel de la aplicación web. A continuación se van a enumerar las mejoras identificadas por los desarrolladores del sistema, y por los usuarios del mismo en sus evaluaciones sobre CumuloNimbo:

- Mejoras de la aplicación web.
    - **Notificaciones**. En el sistema ocurren una serie de acciones que el usuario debe de deducir por como reacciona el sistema. Por ejemplo, el caso de que se inicia una gráfica sin valores para denotar que el componente no está exportando métricas. Se considera que además de esto, el usuario debería de recibir algún tipo de notificación que lo alerte de estas acciones de interés para él. Se podría pensar incluso en un panel de notificaciones con el historial de las más importantes.
    - **Múltiples dashboards**. Para que el usuario pueda por ejemplo emplear cada dashboard para un tipo de métricas. En este momento el usuario puede diferenciar en el panel del dashboard a cada métrica en base a los colores configurables, pero se considera que no es suficiente y sería mucho más sencillo la diferenciación por dashboards.
    - **Selección de métricas desde el dashboard**. Para añadir métricas al dashboard se debe de hacer desde el panel del explorador, navegando entre los recursos y seleccionando las métricas. Se considera que se debería de poder añadir estas métricas desde el propio dashboard, para que los usuarios que ya conozcan perfectamente el sistema monitorizado, no tengan que ir al explorador del sistema.
    - **Dashboard dinámico**. En este momento las gráficas de las métricas se posicionan en el orden en el que son añadidas al panel. Este orden debería de ser configurable, pudiendo mover dichas gráficas y posicionarlas en el lugar

que se desee. Además, los tamaños de dichas gráficas también deberían de ser configurables.

- **Subida y descarga de configuraciones**. Desde la aplicación web se debe poder exportar e importar configuraciones del sistema, para evitar que cada vez que este se inicia halla que estar configurándolo. Estas configuraciones abarcan principalmente a la disposición de un dashboard, o la configuración del sistema a monitorizar.
- **Rendimiento**. Investigar acerca de porqué el rendimiento de la aplicación se ve afectado al tener muchas gráficas en el dashboard, y si éste puede ser subsanado de alguna manera.

- Mejoras del sistema de monitorización.
  - **Configuración desde la aplicación**. Desde la interfaz, se deben de poder seleccionar las métricas que deban de ser monitorizadas para un sesión, y que no se estén empleando recursos en monitorizar todas las configuradas en los ficheros de métricas, malgastando así capacidad de cómputo.
  - **Histórico**. En caso de que haya una métrica del sistema que no se este monitorizando en el dashboard, y se necesite ver que ocurrió con ella en un espacio de tiempo, no es posible. Esto impide poder identificar ciertos comportamientos del sistema. Por lo que es necesario tener una opción para la consulta de los valores de una métrica en un espacio de tiempo pasado.

Las mejoras, como para la mayoría de las cosas, tienden a ser infinitas. En los listados anteriores se han reflejado las que se consideran de mayor prioridad, de cara a futuras mejoras del sistema de monitorización.

Capítulo 7

ANEXOS

En el anexo se encuentra información acerca de como realizar las configuraciones para iniciar el sistema de monitorización. La primera sección 7.1 recoge información acerca de los ficheros de configuración del instalador y de como iniciar el sistema. La segunda sección 7.2 muestra el fichero de configuración adaptado para el despliegue de CumuloNimbo, que se ha empleado para el ejemplo de uso.

7.1. Manual del instalador

Este es un manual en el que se explica como configurar los tres ficheros de configuración del sistema de monitorización y como iniciarlo. Los tres ficheros de configuración se corresponden a la configuración del servidor, del agente y del sistema a monitorizar. Para el inicio del sistema es necesario iniciar un servidor, la aplicación web y el número de agentes necesarios, uno por cada máquina, según el número de máquinas que tengan elementos que monitorizar.

7.1.1. Configuración

Antes de iniciar el servidor se debe de tener en cuenta que los componentes del modelo (HBase y Zookeeper) deben de estar ejecutándose en la misma máquina en la que el servidor se va a ejecutar.

Configurar el servidor El fichero se encuentra dentro de la carpeta del instalador en /etc/config/server-settings.xml.

- **PORT_AGENT**. Puerto del agente. Este puerto debe de ser el mismo en el fichero de configuración del agente y el mismo para todos los agentes que se ejecuten.
- **BASEDIR**. Path en donde se ha generado el instalador.
- **IP_REGISTRY**. IP de la máquina donde el registro se está ejecutando.
- **PORT_REGISTRY**. Puerto del registro.

- **PATH_REGISTRY**. ruta del registro en dónde está almacenada la información del sistema a monitorizar.
- **IP_SERVER**. IP del servidor.
- **PORT_SERVER**. Puerto del servidor. Este debe de ser el mismo que el del fichero de configuración del agente.

Listing 7.1: Configuración

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<serverSettings>
    <agentPort>$PORT_AGENT</agentPort>
    <configurationManagerXMLFileName>$BASEDIR/etc/config/
        Configuration.xml</configurationManagerXMLFileName>
    <configurationManagerXSDFileName>$BASEDIR/etc/config/
        Configuration.xsd</configurationManagerXSDFileName>
    <entityTypeDefinitionXSDFileName>EntityTypeDefinition.xsd</
        entityTypeDefinitionXSDFileName>
    <entityTypesPath>$BASEDIR/etc/entities</entityTypesPath>
    <metricTypesDefinitionXSDFileName>MetricTypesDefinition.xsd</
        metricTypesDefinitionXSDFileName>
    <pullingRate>0</pullingRate>
    <pushingRate>0</pushingRate>
    <recordingRate>1000</recordingRate>
    <registryQuorumAddress>$IP_REGISTRY</registryQuorumAddress>
    <registryQuorumPort>$PORT_REGISTRY</registryQuorumPort>
    <registryRootPath>$PATH_REGISTRY</registryRootPath>
    <registrySessionTimeout>10000</registrySessionTimeout>
    <serverInetAddressString>$IP_SERVER</serverInetAddressString>
    <serverPort>$PORT_SERVER</serverPort>
</serverSettings>
```

Configurar el agente El fichero se encuentra dentro de la carpeta del instalador en /etc/config/agent-settings.xml.

- **IP_AGENT**. IP del agente en donde va a ejecutar.
- **PORT_AGENT**. Puerto del agente. Este puerto debe de ser el mismo que en el fichero del servidor.
- **IP_SERVER**. IP donde se ejecuta el servidor.
- **PORT_SERVER**. Puerto del servidor. Debe de ser el mismo que en el fichero del servidor.

Nota: Para un despliegue de múltiples agentes , todos los agentes deben de tener el mismo puerto y diferente IP.

Listing 7.2: Configuración

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<agentSettings>
    <agentInetAddressString>$IP_AGENT</agentInetAddressString>
    <agentPort>$PORT_AGENT</agentPort>
    <pullingRate>1000</pullingRate>
    <pushingRate>1000</pushingRate>
    <serverInetAddressString>$IP_SERVER</serverInetAddressString>
    <serverPort>$PORT_SERVER</serverPort>
</agentSettings>
```

Configurar el sistema a monitorizar El fichero se encuentra dentro de la carpeta del instalador en /etc/config/Configuration.xml.

- Se tiene que configurar este fichero de manera fiel al despliegue del sistema a monitorizar que se haya realizado.
- entityTypeName y entityTypeVersion. Son referencias a los ficheros XML definidos dentro del directorio (BASEDIR/etc/entities). Dentro de estos XMLS se encuentran las definiciones de las métricas que se van a recolectar par ser monitorizadas. Tener especial cuidado con mantener el format (CollectorType-Entity/Metric-EntityTypeName-EntityTypeVersion.xml) de los ficheros. Los valores del entityTypeName y entityTypeVersion deben de ser excatamente los mismos en el nombre del fichero que en el fichero de configuración.
- Para definir la arquitectura del sistema a monitorizar se debe de definir la jerarquía de nodos lógicos acorde al sistema, sin límite ninguno. Al final de las ramas lógicas se deben de establecer las instancias de los roles del sistema con respecto a los nodos fśicos. Estos nodos físicos también deben de ser añadidos al apartado de *Installation*.
- De los atributos del fichero se debe tener especial cuidado con la IP, el entityTypeName y el entityTypeVersion. Estos deben de ser fieles a la realidad para que el sistema funcione correctamente. Para el resto de atributos como name o instanceID se puede configurar como se quiera.

7.1.2. Despliegue

Se describe a continuación como iniciar al servidor, los agentes y la aplicación.

Iniciar Servidor El servidor debe de ser iniciado en la misma máquina en la que se asignaron la IP y el puerto en el fichero de configuración. El ejecutable de inicio se encuentra dentro de la carpeta del instalador en /bin/runServer

Iniciar Agente/s El agente debe de ser iniciado en la misma máquina en la que se asignaron la IP y el puerto en el fichero de configuración. El ejecutable de inicio se encuentra dentro de la carpeta del instalador en /bin/runAgent

Nota. Iniciar siempre los agentes después de que el servidor haya arrancado y populado la base de datos.
Nota. Si se van a desplegar varios agentes, en el fichero de configuración del agente se puede establecer como IP (localhost), para no tener que modificar el fichero para cada uno de los agentes.

Desplegar la aplicación Para iniciar la aplicación se debe iniciar el servidor de tomcat que por defecto ya contiene la aplicación. El ejectuable de inicio se encuentra dentro de la carpeta del instalador en /apache-tomcat-7.0.47/bin/start

Iniciar la aplicación

1. **Navegador**. Acceder a la URL: localhost:8080/MonitoringSystem
2. **Configuración**. El el panel de configuración de la aplicación rellenar la siguiente información:
   - **RootPath**. Ruta dentro del registro en la que se encuentra la información del sistema a monitorizar.
   - **RegistryIP**. IP de la máquina en la que se está ejecutando el registro.
   - **ReegistryPort**. Puerto del registro.
   - **RegistrySessionTimeOut**. Tiempo de refresco del registro. Habitualmente 10000.

7.2. Fichero de Configuración

A continuación se adjuntan los ficheros XML y XSD del sistema de monitorización. El fichero XML se corresponde al empleado para el caso de uso.

7.2.1. XML

Listing 7.3: Configuración

```
<?xml version="1.0"?>

<Configuration>
    <Installation>
        <AvailableNodes>
                <Node ipAddress="192.168.1.193" entityTypeName="Sigar"
                    entityTypeVersion="1.6.4"/>
            <Node ipAddress="192.168.1.194" entityTypeName="Sigar"
                entityTypeVersion="1.6.4"/>
                <Node ipAddress="192.168.1.195" entityTypeName="Sigar"
                    entityTypeVersion="1.6.4"/>
            <Node ipAddress="192.168.1.196" entityTypeName="Sigar"
                entityTypeVersion="1.6.4"/>
            <Node ipAddress="192.168.1.201" entityTypeName="Sigar"
                entityTypeVersion="1.6.4"/>
            <Node ipAddress="192.168.1.202" entityTypeName="Sigar"
                entityTypeVersion="1.6.4"/>
            <Node ipAddress="192.168.1.204" entityTypeName="Sigar"
                entityTypeVersion="1.6.4"/>
            <Node ipAddress="192.168.1.205" entityTypeName="Sigar"
                entityTypeVersion="1.6.4"/>
            <Node ipAddress="192.168.1.206" entityTypeName="Sigar"
                entityTypeVersion="1.6.4"/>
            <Node ipAddress="192.168.1.207" entityTypeName="Sigar"
                entityTypeVersion="1.6.4"/>
            <Node ipAddress="192.168.1.208" entityTypeName="Sigar"
                entityTypeVersion="1.6.4"/>
            <Node ipAddress="192.168.1.209" entityTypeName="Sigar"
                entityTypeVersion="1.6.4"/>
            <Node ipAddress="192.168.1.210" entityTypeName="Sigar"
                entityTypeVersion="1.6.4"/>
            <Node ipAddress="192.168.1.211" entityTypeName="Sigar"
                entityTypeVersion="1.6.4"/>
        </AvailableNodes>
        <AllocatedNodes>
        </AllocatedNodes>
    </Installation>
    <Deployment>
        <LogicalResource name="HadoopHDFS">
            <PropertyList>
            </PropertyList>
            <ResourceList>
                <LogicalResource name="NameNode">
                    <PropertyList>
```

```
31                </PropertyList>
32                <ResourceList>
33                   <PhysicalResource name="192.168.1.207" ipAddress=
                        "192.168.1.207" instanceID="/HadoopHDFS/
                        NameNode/192.168.1.207" entityTypeName="
                        HadoopHDFSNameNode" entityTypeVersion="2.0.0">
34                      <PropertyList>
35                      </PropertyList>
36                   </PhysicalResource>
37                </ResourceList>
38             </LogicalResource>
39             <LogicalResource name="DataNode">
40                <PropertyList>
41                </PropertyList>
42                <ResourceList>
43                   <PhysicalResource name="192.168.1.201" ipAddress=
                        "192.168.1.201" instanceID="/HadoopHDFS/
                        DataNode/192.168.1.201" entityTypeName="
                        HadoopHDFSDataNode" entityTypeVersion="2.0.0">
44                      <PropertyList>
45                      </PropertyList>
46                   </PhysicalResource>
47                   <PhysicalResource name="192.168.1.202" ipAddress=
                        "192.168.1.202" instanceID="/HadoopHDFS/
                        DataNode/192.168.1.202" entityTypeName="
                        HadoopHDFSDataNode" entityTypeVersion="2.0.0">
48                      <PropertyList>
49                      </PropertyList>
50                   </PhysicalResource>
51                   <PhysicalResource name="192.168.1.204" ipAddress=
                        "192.168.1.204" instanceID="/HadoopHDFS/
                        DataNode/192.168.1.204" entityTypeName="
                        HadoopHDFSDataNode" entityTypeVersion="2.0.0">
52                      <PropertyList>
53                      </PropertyList>
54                   </PhysicalResource>
55                   <PhysicalResource name="192.168.1.205" ipAddress=
                        "192.168.1.205" instanceID="/HadoopHDFS/
                        DataNode/192.168.1.205" entityTypeName="
                        HadoopHDFSDataNode" entityTypeVersion="2.0.0">
56                      <PropertyList>
57                      </PropertyList>
58                   </PhysicalResource>
59                   <PhysicalResource name="192.168.1.206" ipAddress=
                        "192.168.1.206" instanceID="/HadoopHDFS/
                        DataNode/192.168.1.206" entityTypeName="
                        HadoopHDFSDataNode" entityTypeVersion="2.0.0">
60                      <PropertyList>
61                      </PropertyList>
62                   </PhysicalResource>
63                   <PhysicalResource name="192.168.1.208" ipAddress=
                        "192.168.1.208" instanceID="/HadoopHDFS/
                        DataNode/192.168.1.208" entityTypeName="
                        HadoopHDFSDataNode" entityTypeVersion="2.0.0">
```

```
64                      <PropertyList>
65                      </PropertyList>
66                  </PhysicalResource>
67                  <PhysicalResource name="192.168.1.209" ipAddress=
                        "192.168.1.209" instanceID="/HadoopHDFS/
                        DataNode/192.168.1.209" entityTypeName="
                        HadoopHDFSDataNode" entityTypeVersion="2.0.0">
68                      <PropertyList>
69                      </PropertyList>
70                  </PhysicalResource>
71                  <PhysicalResource name="192.168.1.211" ipAddress=
                        "192.168.1.211" instanceID="/HadoopHDFS/
                        DataNode/192.168.1.211" entityTypeName="
                        HadoopHDFSDataNode" entityTypeVersion="2.0.0">
72                      <PropertyList>
73                      </PropertyList>
74                  </PhysicalResource>
75              </ResourceList>
76          </LogicalResource>
77      </ResourceList>
78  </LogicalResource>
79  <LogicalResource name="HBase">
80      <PropertyList>
81      </PropertyList>
82      <ResourceList>
83          <LogicalResource name="Master">
84              <PropertyList>
85              </PropertyList>
86              <ResourceList>
87                  <PhysicalResource name="192.168.1.207" ipAddress=
                        "192.168.1.207" instanceID="/HBase/Master
                        /192.168.1.207" entityTypeName="HBaseMaster"
                        entityTypeVersion="0.94.6">
88                      <PropertyList>
89                      </PropertyList>
90                  </PhysicalResource>
91              </ResourceList>
92          </LogicalResource>
93          <LogicalResource name="RegionServer">
94              <PropertyList>
95              </PropertyList>
96              <ResourceList>
97                  <PhysicalResource name="192.168.1.201" ipAddress=
                        "192.168.1.201" instanceID="/HBase/
                        RegionServer/192.168.1.201" entityTypeName="
                        HBaseRegionServer" entityTypeVersion="0.94.6">
98                      <PropertyList>
99                      </PropertyList>
100                 </PhysicalResource>
101                 <PhysicalResource name="192.168.1.202" ipAddress=
                        "192.168.1.202" instanceID="/HBase/
                        RegionServer/192.168.1.202" entityTypeName="
                        HBaseRegionServer" entityTypeVersion="0.94.6">
102                     <PropertyList>
```

```
103                     </PropertyList>
104                 </PhysicalResource>
105                 <PhysicalResource name="192.168.1.204" ipAddress=
                        "192.168.1.204" instanceID="/HBase/
                        RegionServer/192.168.1.204" entityTypeName="
                        HBaseRegionServer" entityTypeVersion="0.94.6">
106                     <PropertyList>
107                     </PropertyList>
108                 </PhysicalResource>
109                 <PhysicalResource name="192.168.1.205" ipAddress=
                        "192.168.1.205" instanceID="/HBase/
                        RegionServer/192.168.1.205" entityTypeName="
                        HBaseRegionServer" entityTypeVersion="0.94.6">
110                     <PropertyList>
111                     </PropertyList>
112                 </PhysicalResource>
113                 <PhysicalResource name="192.168.1.206" ipAddress=
                        "192.168.1.206" instanceID="/HBase/
                        RegionServer/192.168.1.206" entityTypeName="
                        HBaseRegionServer" entityTypeVersion="0.94.6">
114                     <PropertyList>
115                     </PropertyList>
116                 </PhysicalResource>
117                 <PhysicalResource name="192.168.1.208" ipAddress=
                        "192.168.1.208" instanceID="/HBase/
                        RegionServer/192.168.1.208" entityTypeName="
                        HBaseRegionServer" entityTypeVersion="0.94.6">
118                     <PropertyList>
119                     </PropertyList>
120                 </PhysicalResource>
121                     <PhysicalResource name="192.168.1.209"
                            ipAddress="192.168.1.209" instanceID="/
                            HBase/RegionServer/192.168.1.209"
                            entityTypeName="HBaseRegionServer"
                            entityTypeVersion="0.94.6">
122                     <PropertyList>
123                     </PropertyList>
124                 </PhysicalResource>
125                 <PhysicalResource name="192.168.1.211" ipAddress=
                        "192.168.1.211" instanceID="/HBase/
                        RegionServer/192.168.1.211" entityTypeName="
                        HBaseRegionServer" entityTypeVersion="0.94.6">
126                     <PropertyList>
127                     </PropertyList>
128                 </PhysicalResource>
129             </ResourceList>
130         </LogicalResource>
131     </ResourceList>
132 </LogicalResource>
133 <LogicalResource name="ZooKeeper">
134     <PropertyList>
135     </PropertyList>
136     <ResourceList>
137         <LogicalResource name="Standalone">
```

```
138                 <PropertyList>
139                 </PropertyList>
140                 <ResourceList>
141                     <PhysicalResource name="192.168.1.207" ipAddress=
                            "192.168.1.207" instanceID="/ZooKeeper/Quorum
                            /192.168.1.207" entityTypeName="
                            ZooKeeperStandalone" entityTypeVersion="3.4.5"
                            >
142                         <PropertyList>
143                         </PropertyList>
144                     </PhysicalResource>
145                 </ResourceList>
146             </LogicalResource>
147         </ResourceList>
148     </LogicalResource>
149     <LogicalResource name="Escada">
150         <PropertyList>
151         </PropertyList>
152         <ResourceList>
153             <PhysicalResource name="192.168.1.196" ipAddress="
                    192.168.1.196" instanceID="/Escada/192.168.1.196"
                    entityTypeName="EscadaTPCC" entityTypeVersion="0.1">
154                 <PropertyList>
155                 </PropertyList>
156             </PhysicalResource>
157                  <PhysicalResource name="192.168.1.189" ipAddress="
                        192.168.1.189" instanceID="/Escada
                        /192.168.1.189" entityTypeName="EscadaTPCC"
                        entityTypeVersion="0.1">
158                         <PropertyList>
159                         </PropertyList>
160                  </PhysicalResource>
161                 <PhysicalResource name="192.168.1.186" ipAddress="
                        192.168.1.186" instanceID="/Escada/192.168.1.186"
                         entityTypeName="EscadaTPCC" entityTypeVersion="
                        0.1">
162                         <PropertyList>
163                         </PropertyList>
164                  </PhysicalResource>
165                  <PhysicalResource name="192.168.1.187" ipAddress="
                         192.168.1.187" instanceID="/Escada/192.168.1.187
                        " entityTypeName="EscadaTPCC" entityTypeVersion=
                        "0.1">
166                         <PropertyList>
167                         </PropertyList>
168                  </PhysicalResource>
169         </ResourceList>
170     </LogicalResource>
171   <LogicalResource name="EscadaOLAP">
172     <PropertyList>
173     </PropertyList>
174     <ResourceList>
175         <PhysicalResource name="192.168.1.178" ipAddress="
                192.168.1.178" instanceID="/EscadaOLAP/192.168.1.178"
```

```
                entityTypeName="EscadaTPCCOLAP" entityTypeVersion="0.1"
                >
176           <PropertyList>
177           </PropertyList>
178         </PhysicalResource>
179         <PhysicalResource name="192.168.1.179" ipAddress="
                192.168.1.179" instanceID="/EscadaOLAP/192.168.1.179"
                entityTypeName="EscadaTPCCOLAP" entityTypeVersion="0.1">
180           <PropertyList>
181           </PropertyList>
182         </PhysicalResource>
183         <PhysicalResource name="192.168.1.180" ipAddress="
                192.168.1.180" instanceID="/EscadaOLAP/192.168.1.180"
                entityTypeName="EscadaTPCCOLAP" entityTypeVersion="0.1">
184           <PropertyList>
185           </PropertyList>
186         </PhysicalResource>
187         <PhysicalResource name="192.168.1.181" ipAddress="
                192.168.1.181" instanceID="/EscadaOLAP/192.168.1.181"
                entityTypeName="EscadaTPCCOLAP" entityTypeVersion="0.1">
188           <PropertyList>
189           </PropertyList>
190         </PhysicalResource>
191       </ResourceList>
192     </LogicalResource>
193       <LogicalResource name="QueryEngine">
194           <PropertyList>
195           </PropertyList>
196           <ResourceList>
197               <PhysicalResource name="192.168.1.207" ipAddress="
                      192.168.1.207" instanceID="/QueryEngine
                      /192.168.1.207" entityTypeName="DbDerby+QueryEngine"
                       entityTypeVersion="0.4">
198                   <PropertyList>
199                   </PropertyList>
200               </PhysicalResource>
201               <PhysicalResource name="192.168.1.210" ipAddress="
                      192.168.1.210" instanceID="/QueryEngine
                      /192.168.1.210" entityTypeName="DbDerby+QueryEngine"
                       entityTypeVersion="0.4">
202                   <PropertyList>
203                   </PropertyList>
204               </PhysicalResource>
205           </ResourceList>
206       </LogicalResource>
207       <LogicalResource name="Transactions">
208           <PropertyList>
209           </PropertyList>
210           <ResourceList>
211               <LogicalResource name="Logger">
212                   <PropertyList>
213                   </PropertyList>
214                   <ResourceList>
```

```
215             <PhysicalResource name="192.168.1.207" ipAddress="
                    192.168.1.207" instanceID="/QueryEngine
                    /192.168.1.207">
216                 <PropertyList>
217                 </PropertyList>
218             </PhysicalResource>
219             <PhysicalResource name="192.168.1.210" ipAddress="
                    192.168.1.210" instanceID="/QueryEngine
                    /192.168.1.210">
220                 <PropertyList>
221                 </PropertyList>
222             </PhysicalResource>
223                 </ResourceList>
224             </LogicalResource>
225             <LogicalResource name="LocalTransactionManager">
226                 <PropertyList>
227                 </PropertyList>
228                 <ResourceList>
229             <PhysicalResource name="192.168.1.207" ipAddress="
                    192.168.1.207" instanceID="/QueryEngine
                    /192.168.1.207">
230                 <PropertyList>
231                 </PropertyList>
232             </PhysicalResource>
233             <PhysicalResource name="192.168.1.210" ipAddress="
                    192.168.1.210" instanceID="/QueryEngine
                    /192.168.1.210">
234                 <PropertyList>
235                 </PropertyList>
236             </PhysicalResource>
237                 </ResourceList>
238             </LogicalResource>
239             <LogicalResource name="ConflictManager">
240                 <PropertyList>
241                 </PropertyList>
242                 <ResourceList>
243                     <PhysicalResource name="192.168.1.195" ipAddress=
                            "192.168.1.195" instanceID="/Transactions/
                            ConflictManager/192.168.1.195" entityTypeName=
                            "ConflictManager" entityTypeVersion="0.4">
244                         <PropertyList>
245                         </PropertyList>
246                     </PhysicalResource>
247                 </ResourceList>
248             </LogicalResource>
249             <LogicalResource name="ConfigurationManager">
250                 <PropertyList>
251                 </PropertyList>
252                 <ResourceList>
253                     <PhysicalResource name="192.168.1.194" ipAddress=
                            "192.168.1.194" instanceID="/Transactions/
                            ConfigurationManager/192.168.1.194">
254                         <PropertyList>
255                         </PropertyList>
```

```
256                    </PhysicalResource>
257                 </ResourceList>
258              </LogicalResource>
259              <LogicalResource name="CommitSequencer">
260                 <PropertyList>
261                 </PropertyList>
262                 <ResourceList>
263                    <PhysicalResource name="192.168.1.194" ipAddress=
                          "192.168.1.194" instanceID="/Transactions/
                          CommitSequencer/192.168.1.194" entityTypeName=
                          "CommitSequencer" entityTypeVersion="0.4">
264                       <PropertyList>
265                       </PropertyList>
266                    </PhysicalResource>
267                 </ResourceList>
268              </LogicalResource>
269              <LogicalResource name="SnapshotServer">
270                 <PropertyList>
271                 </PropertyList>
272                 <ResourceList>
273                    <PhysicalResource name="192.168.1.194" ipAddress=
                          "192.168.1.194" instanceID="/Transactions/
                          SnapshotServer/192.168.1.194" entityTypeName="
                          SnapshotServer" entityTypeVersion="0.4">
274                       <PropertyList>
275                       </PropertyList>
276                    </PhysicalResource>
277                 </ResourceList>
278              </LogicalResource>
279           </ResourceList>
280        </LogicalResource>
281     </Deployment>
282 </Configuration>
```

7.2.2. XSD

Listing 7.4: Definición

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0
    ">

    <xs:element name="Configuration" type="Configuration"/>

    <xs:complexType name="Configuration">
        <xs:sequence>
            <xs:element name="Installation" type="Installation"/>
            <xs:element name="Deployment" type="ResourceList"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="Installation">
        <xs:sequence>
            <xs:element name="AvailableNodes" type="NodeList"/>
            <xs:element name="AllocatedNodes" type="NodeList"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="NodeList">
        <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element name="Node" type="Node"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="Node">
        <xs:attribute name="entityTypeName" type="xs:string"/>
        <xs:attribute name="entityTypeVersion" type="xs:string"/>
        <xs:attribute name="ipAddress" type="xs:string" use="required
            "/>
    </xs:complexType>

    <xs:complexType name="ResourceList">
        <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:choice>
                <xs:element name="LogicalResource" type="
                    LogicalResource"/>
                <xs:element name="PhysicalResource" type="
                    PhysicalResource"/>
            </xs:choice>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="LogicalResource">
        <xs:sequence>
            <xs:element name="PropertyList" type="PropertyList"/>
            <xs:element name="ResourceList" type="ResourceList"/>
        </xs:sequence>
        <xs:attribute name="name" type="xs:string" use="required"/>
```

```
47        <xs:attribute name="entityTypeName" type="xs:string"/>
48        <xs:attribute name="entityTypeVersion" type="xs:string"/>
49    </xs:complexType>
50
51    <xs:complexType name="PhysicalResource">
52        <xs:sequence>
53              <xs:element name="PropertyList" type="PropertyList"/>
54        </xs:sequence>
55        <xs:attribute name="name" type="xs:string" use="required"/>
56        <xs:attribute name="ipAddress" type="xs:string" use="required
              "/>
57        <xs:attribute name="instanceID" type="xs:string" use="
              required"/>
58        <xs:attribute name="entityTypeName" type="xs:string"/>
59        <xs:attribute name="entityTypeVersion" type="xs:string"/>
60    </xs:complexType>
61
62    <xs:complexType name="PropertyList">
63          <xs:sequence minOccurs="0" maxOccurs="unbounded">
64          <xs:element name="Property" type="Property"/>
65          </xs:sequence>
66    </xs:complexType>
67
68    <xs:complexType name="Property">
69        <xs:attribute name="name" type="xs:string" use="required"/>
70        <xs:attribute name="value" type="xs:string" use="required"/>
71    </xs:complexType>
72
73 </xs:schema>
```

Bibliografía

[1] Lars George. *HBase: The definitive Guide*. O'Reilly Media, Inc, 2011.

[2] Google. Gwt developers guide. http://www.gwtproject.org/doc/latest/DevGuide.html.

[3] Marko Grönroos. Vaadin mvp. https://vaadin.com/web/magi/home/-/blogs/model-view-presenter-pattern-with-vaadin, September 2013.

[4] Marko Grönroos. *Book of Vaadin*. Vaadin Ltd, vaadin 7 edition - 3rd revision edition, 2014.

[5] Ricardo Jimenez-Peris and Others. A highly scalable transactional multi-tier platform as a service. Technical report, Distributed Systems Lab(LSD), Univ. Politecnica de Madrid(UPM), 2013.

[6] Ricardo Jiménez-Peris, Marta Patiño-Martínez, and Iván Brondino. Cumulonimbo: Procesamiento transaccional distribuido en paralelo. Technical report, Distributed Systems Lab(LSD), Univ. Politecnica de Madrid(UPM), 2012.

[7] Flavio Junqueira and Benjamin Reed. *ZooKeeper*. O'Reilly Media, Inc, 2013.

[8] Jeff Knupp. What is a web framework. http://jeffknupp.com/blog/2014/03/03/what-is-a-web-framework/, March 2014.

[9] Anthony Wing Kosner. The appification of everything will transform the world's 360 million web sites. *Forbes*, December 2012.

[10] Sergio Luján Mora. *Programación de aplicaciones web: historia, principios básicos y clientes web*. Universidad de Alicante, Octubre 2002.

[11] Addy Osmani. *Learning JavaScript Design Patterns*, chapter JavaScript MV* Patterns. O'Reilly Media, Inc, 2012.

[12] RebelLabs. The curious coder's java web frameworks comparaison. Technical report, ZEROTURNAROUND, July 2013.

[13] Trygve Reenskaug. The original mvc reports. Technical report, Dept. of informatics, Universuty of Oslo, 1979.

[14] VMware. Introduction to hyperic monitoring.

[15] Wikipedia. Comparaison of web application frameworks.

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
	Fecha/Hora	Fri Jun 06 19:39:59 CEST 2014
	Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
	Numero de Serie	630
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)